SUPERVISED GRAPH LEARNING

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application claims the benefit of and priority to U.S. Provisional Application No. 63/244,781, filed September 16, 2021, which is hereby incorporated by reference herein in its entirety.

TECHNICAL FIELD

[0002] The present disclosure relates to machine learning and, more particularly, to supervised graph learning that creates a probabilistic graph, without any predetermined graph structure, using training data.

BACKGROUND

[0003] A graph is a data structure that includes nodes and edges. There are many different types of graphs, such as undirected graphs, directed graphs, hypergraphs, graphs with self-edges, graphs without self-edges, and/or bipartite graphs, among others. Graphs are applicable to widespread numbers and types of applications, including classification tasks, for example.

SUMMARY

[0004] The present disclosure relates to supervised graph learning that creates a probabilistic graph, without any predetermined graph structure, using training data. The present disclosure provides an improvement to technology in the machine learning field and creates graph models that have no predetermined structure prior to the training. The graph models created by the disclosed technology can be applied to new observed data to make predictions/inferences. Unless indicated otherwise by context, the terms “inference” and “prediction” may be used interchangeably herein.

[0005] In accordance with aspects of the present disclosure, a system includes one or more processors and at least one memory storing instructions. The instructions, when executed by the one or more processors, cause the system to access training data relating to a plurality of variables, determine a factor graph model based on the training data where the factor graph model includes component factors, estimate probability density functions for the component factors based on Monte Carlo integration in frequency domain, and provide the factor graph model with the estimated probability density functions for the component factors.

[0006] In various embodiments of the system, in determining the factor graph model, the instructions, when executed by the one or more processors, cause the system to determine low-order moments based on the training data.

[0007] In various embodiments of the system, in determining the factor graph model, the instructions, when executed by the one or more processors, cause the system to identify subsets of dependent variables based on the low-order moments.

[0008] In various embodiments of the system, in identifying the subsets of dependent variables, the instructions, when executed by the one or more processors, cause the system to compare whether joint variable moments deviate from products of individual variable moments.

[0009] In various embodiments of the system, in determining the factor graph model, the instructions, when executed by the one or more processors, cause the system to create factor nodes corresponding to the subsets of dependent variables and adding edges from the factor nodes to corresponding variable nodes.

[0010] In various embodiments of the system, in estimating probability density functions for the component factors, the instructions, when executed by the one or more processors, cause the

system to estimate functional forms of the probability density functions for the component factors using Fourier domain synthesis.

[0011] In various embodiments of the system, in estimating probability density functions for the component factors, the instructions, when executed by the one or more processors, cause the system to estimate the probability density functions for the component factors using the functional forms, joint variable moments, and direct Monte Carlo integration in the frequency domain.

[0012] In various embodiments of the system, the instructions, when executed by the one or more processors, further cause the system to apply the factor graph model with the estimated probability density functions for the component factors to observed variables to make an inference.

[0013] In accordance with aspects of the present disclosure, a method includes accessing training data relating to a plurality of variables, determining a factor graph model based on the training data where the factor graph model includes component factors, estimating probability density functions for the component factors based on Monte Carlo integration in frequency domain, and providing the factor graph model with the estimated probability density functions for the component factors.

[0014] In various embodiments of the method, determining the factor graph model includes determining low-order moments based on the training data.

[0015] In various embodiments of the method, determining the factor graph model includes identifying subsets of dependent variables based on the low-order moments.

[0016] In various embodiments of the method, identifying the subsets of dependent variables includes comparing whether joint variable moments deviate from products of individual variable moments.

[0017] In various embodiments of the method, determining the factor graph model includes creating factor nodes corresponding to the subsets of dependent variables and adding edges from the factor nodes to corresponding variable nodes.

[0018] In various embodiments of the method, estimating the probability density functions for the component factors includes estimating functional forms of the probability density functions for the component factors using Fourier domain synthesis.

[0019] In various embodiments of the method, estimating the probability density functions for the component factors includes estimating the probability density functions for the component factors using the functional forms, joint variable moments, and direct Monte Carlo integration in the frequency domain.

[0020] In various embodiments of the method, the method includes applying the factor graph model with the estimated probability density functions for the component factors to observed variables to make an inference.

[0021] In accordance with aspects of the present disclosure, a non-transitory computer-readable medium stores instructions which, when executed by a computer, causes the computer to perform a method that includes accessing training data relating to a plurality of variables, determining a factor graph model based on the training data where the factor graph model includes component factors, estimating probability density functions for the component factors based on Monte Carlo integration in the frequency domain, and providing the factor graph model with the estimated probability density functions for the component factors.

[0022] In accordance with aspects of the present disclosure, a method includes: accessing training data relating to a plurality of variables; determining a probabilistic graph model based on the training data, where the probabilistic graph model includes nodes and edges deduced from the

training data and where the probabilistic graph model has no predetermined structure prior to the nodes and edges being deduced; estimating probabilistic parameters of the probabilistic graph model using the training data; and applying the probabilistic graph model with the estimated probabilistic parameters to new observed data to make a prediction.

[0023] In various embodiments of the method, the probabilistic graph model is a factor graph model.

[0024] Further details and aspects of exemplary embodiments of the present disclosure are described in more detail below.

BRIEF DESCRIPTION OF THE DRAWINGS

[0025] FIG. 1 is a flow diagram of an exemplary operation, in accordance with aspects of the present disclosure;

[0026] FIG. 2 is a diagram of an exemplary factor graph corresponding to an error correction system, in accordance with aspects of the present disclosure;

[0027] FIG. 3 is a diagram of exemplary matrices relating to the error correction system, in accordance with aspects of the present disclosure;

[0028] FIG. 4 is a diagram of exemplary computations relating to sum-product algorithm, in accordance with aspects of the present disclosure;

[0029] FIG. 5 is a diagram of further exemplary computations relating to sum-product algorithm, in accordance with aspects of the present disclosure;

[0030] FIG. 6 is a diagram of exemplary bit error rate curves, in accordance with aspects of the present disclosure;

[0031] FIG. 7 is a diagram of exemplary parameters relating to neural network implementations, in accordance with aspects of the present disclosure; and

[0032] FIG. 8 is a block diagram of exemplary components of a computing system, in accordance with aspects of the present disclosure.

DETAILED DESCRIPTION

[0033] The present disclosure relates to supervised graph learning that creates a probabilistic graph, without any predetermined graph structure, using training data. The present disclosure provides an improvement to technology in the machine learning field and creates graph models that have no predetermined structure prior to the training. The graph models created by the disclosed technology can be applied to new observed data to make predictions/inferences.

[0034] A system may be characterized by variables. For example, a system for predicting temperature may have variables such as location (e.g., longitude and latitude), time of year (e.g., number of days since start of the year), time of day, weather conditions (e.g., sunny, rainy, etc.), and traffic congestion level, among other variables. Systems for other applications have other variables. In various embodiments, a system variable may be a variable that is observable (e.g., measured by a sensor) or a variable that is not observable but may be computed, e.g., a variable whose value is to be predicted.

[0035] As mentioned above, a graph is a data structure that includes nodes and edges. As used herein, a "probabilistic graph" is a graph that models statistical relationships within a system of variables, such as statistical relationships between system inputs and system outputs or statistical relationships between variables or subgroups of variables of the system.

[0036] One example of a probabilistic graph is a factor graph. A factor graph is a bipartite graph, in which (a) there are two-types of nodes: variable nodes and factor nodes, and (b) edges of the graph connect only to nodes of different type (i.e., edges connect only variable nodes to factor nodes). The variable nodes represent the variables of the system, and the factor nodes represent

the component factors (local statistical dependence) of the global system. 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.

[0037] The disclosed technology is generally applicable to probabilistic graphs. Factor graphs are used as the primary example, but it is intended that any disclosure herein relating to factor graphs may be applied to probabilistic graphs.

[0038] The present disclosure improves the technology of the information theory, artificial intelligence, and machine learning disciplines. In accordance with aspects of the present disclosure, the disclosed technology realizes a generalized graph-based framework for automated learning in intelligent systems. Aspects of the present disclosure provide automatic graph learning framework employs factor graphs to represent any stochastic system of variables and factorized realizations of their joint probability density function. Algorithms are disclosed that are capable of learning statistical relationships between system variables, which, in various embodiments, involves constructing an appropriate factor graph representation and generating estimates of its component probability density functions, from training data.

[0039] As used herein, the terms “automatic graph learning” and “supervised graph learning” may be used interchangeably to refer to machine learning that uses training data to create a probabilistic graph that represents a stochastic system of variables, including factor graphs which represent factorized realizations of joint probability density functions of system variables. Unlike machine learning using neural networks (NN), which trains a predetermined network having a predetermined structure, the automatic graph learning of the present disclosure does not start with any predetermined graph structure. Although graphical structure is a basic characteristic of neural networks, 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 and recurrent neural networks 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 any learning system. Unlike neural networks, and as described in more detail below, the probabilistic graphs created by the automatic graph learning disclosed herein can result in any structure deduced from the training data.

[0040] Aspects of the disclosed technology use probability and information theory for modeling stochastic systems and implementing probabilistic inference algorithms and use supervised learning, i.e., training-based learning from principles of probability theory. As described above, a graph model for the system in question is *a priori* unknown, and training data is used to deduce an appropriate probabilistic graph model and, in the case of factor graphs, functional approximation of its component factors. The developed methodology is general in the sense that it can be applied to any data system, wherein a suitable probabilistic graph model is automatically learned from training data and inference computations are performed on the learned graph. In the case of factor graph system models, and given a characterization of is component factors, an instance of the sum-product algorithm can be used to compute marginal densities, such as the *a posteriori* probabilities, on non-observed system variables.

[0041] In accordance with aspects of the present disclosure, the disclosed technology provides a generalized learning framework that captures statistical structure between a system of variables. Using factor graphs as an example, the disclosed automatic graph learning/supervised graph learning 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.

[0042] In contrast to techniques which select the best model from a predefined set of candidate models, the present disclosure builds a unified framework in which an appropriate probabilistic graph model for representing and processing inference for a system of data is automatically learned from training data. As explained in more detail later herein, the disclosed technology imposes no constraint on the family of densities used to model the posterior density, factor densities, or other densities that arise in belief propagation. Furthermore, as explained later herein, instead of truncating the Fourier series of factor density functions, the disclosed technology uses the relationship of Fourier coefficients to statistical moments to form estimates of the component factors based on truncated orders of moments. As persons skilled in the art will understand, statistical moments are a set of parameters that measure a probability distribution, such as the first four moments: mean, variance, skew and kurtosis. Higher order moments, as well as joint moments involving multiple random variables, are utilized in this disclosure for completely characterizing the joint probability distribution function of subsystems of random variables.

[0043] The disclosed technology further leverages direct Monte Carlo integration of the joint density function in the Fourier domain. The disclosed technology also uses training data to estimate statistical moments via Monte Carlo integration, and the methods are more general because they do not assume that the output is in the form of statistical expectation. Rather, the disclosed technology estimates the density functions directly in the Fourier domain and uses the relationship of the Fourier coefficients to statistical moments. Accordingly, the present disclosure applies a statistical moments-based approach for constructing probabilistic graphical models.

[0044] In accordance with aspects of the present disclosure, the disclosed technology enables learning a factor graph model representative of statistical relationships between a system of variables. In various embodiments, a factor graph model can be learned by using estimates (from training data) of low-order statistical moments. For example, subsets of variables of a system may be tested for dependence, based on the estimated moments from training data, by checking whether their joint moments deviate from the product of their individual moments. Dependent subsets can be selected to construct a factor graph for the system of variables. Graphically, this amounts to creating factor nodes corresponding to the selected subsets of variables and adding edges from the factor nodes to their corresponding variable nodes, to thereby create a factor graph.

[0045] In accordance with aspects of the present disclosure, after a factor graph model is learned, the disclosed technology may determine probability density functions of the component factors. In various embodiments, estimates of component factor probability density functions may be generated using Fourier domain representations and Monte Carlo integration. Fourier coefficients have a relationship to statistical moments. Because component factors were determined using low-order statistical moments, and statistical moments have a relationship with Fourier coefficients, Fourier domain synthesis may be used to estimate the functional form of the component factor probability density functions. Using the Fourier transform synthesis equation, estimates of the component factor probability density functions can be determined using estimates of the joint statistical moments between the system variables or by leveraging direct Monte Carlo integration of the joint density function in the Fourier domain, as described in subsequent sections.

[0046] The result of such operations are a learned factor graph model with estimated probability density functions for the component factors. Because the factor graph model is learned

from training data, the operations of the present disclosure are referred to herein as "supervised graph learning."

[0047] Accordingly, supervised graph learning provides a factor graph model with estimated probability density functions for the component factors. To apply the learned factor graph model to observed variables to make an inference, an instance of the sum-product algorithm can be implemented to process the observed variables and make an inference.

[0048] Accordingly, supervised graph learning and using the sum-product algorithm to apply a learned factor graph to observed variables are disclosed herein.

[0049] The disclosed technology uses training data to learn a factor graph model of the data system and develops estimates of the factor probability density functions (PDFs). A "probability density function" is a term of art. As persons skilled in the art will understand, a probability density function is non-negative everywhere, and its integral over the entire space is equal to one. A PDF can be used to specify the probability of the random variable falling within a particular range of values. As explained below, the estimates of the factor density functions are built using estimates of the joint statistical moments between the system variables and Monte Carlo integration. The disclosed techniques leverage the theorem below, which is described using continuous-time Fourier transforms. Persons skilled in the art will understand how to apply the disclosed techniques using discrete-time Fourier transforms and Fast Fourier Transforms, among other Fourier transform implementations.

[0050] Theorem 1

[0051] The joint PDF of X and Y, denoted $f_{XY}(x, y)$, is uniquely determined by its joint statistical moments, $E[X^m Y^n]$, for all $m = 1,2,3 \ldots$ and $n = 1,2,3, \ldots$.

[0052] Proof

[0053] The theorem follows from substituting the Taylor expansion of the complex exponential in the multivariate Fourier transform of the joint PDF $f_{XY}(x,y)$.

[0054] The PDF of X and Y can be expressed as:

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

EQUATION 1

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

$$F_{XY}(\omega_1,\omega_2) = \frac{1}{(2\pi)^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-i2\pi(\omega_1 x+\omega_2 y)}\,dxdy = \frac{1}{(2\pi)^2}E\left[e^{-i2\pi(\omega_1 X+\omega_2 Y)}\right]$$

$$= \frac{1}{(2\pi)^2}\sum_{m=0}^{\infty}\sum_{n=0}^{\infty}(-i2\pi)^{m+n}\frac{\omega_1^m}{m!}\frac{\omega_2^n}{n!}E[X^m Y^n].$$

EQUATION 2

Hence, assuming convergence of the integrals in Equation 1 and Equation 2, the PDF $f_{XY}(x,y)$ is expressed in terms of its joint moments $E[X^m Y^n]$ for all $m > 0$ and $n > 0$.

[0055] Corollary 1

[0056] The random variables X and Y are independent if and only if the following equation holds for all values of $m > 0$ and $n > 0$:

$$E[X^m Y^n] = E[X^m]E[Y^n].$$

EQUATION 3

[0057] Proof

[0058] 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:

$$E[X^m Y^n] = \frac{(2\pi)^2}{(-i2\pi)^{m+n}} \frac{\partial^{m+n} F_{XY}(\omega_1, \omega_2)}{\partial^m \omega_1 \, \partial^n \omega_2}\bigg|_{\omega_1=0, \omega_2=0}$$

$$= \frac{(2\pi)^2}{(-i2\pi)^{m+n}} \frac{d^m F_X(\omega_1)}{d^m \omega_1}\bigg|_{\omega_1=0} \frac{d^n F_Y(\omega_2)}{d^n \omega_2}\bigg|_{\omega_2=0}$$

$$= E[X^m]E[Y^n],$$

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

[0059] Conversely, substituting $E[X^m Y^n] = E[X^m]E[Y^n]$ in Equation 2 yields $F_{XY}(\omega_1, \omega_2) = F_X(\omega_1)F_Y(\omega_2)$ and thus $f_{XY}(x, y) = f_X(x)f_Y(y)$.

[0060] The above results extend to larger multivariate PDFs, wherein the joint statistical moments must be taken between all combinations of exponents and variables. A result of Corollary 1 is that if Equation 3 does not hold true for any m and n, then X and Y must be dependent random variables—a property that may be used to check for dependence among subsets of system variables.

[0061] In accordance with aspects of the present disclosure, statistical moments estimated from training data are used to select dependent subsets of system variables and, thus, a factorization of the system PDF, as in Equation 4 below. Equation 2 can then be used to estimate the functional form of the component factors. Assuming a stationary system (joint statistics are time-invariant), and independent identically distributed (IID) training data (x_i, y_i), $i = 1, \ldots, T$, we have the following empirical estimates for the joint moments of X and Y:

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

which follows from the law of large numbers. In various embodiments, reduced complexity estimates can be obtained by using only low order, low degree moments to select factors and to estimate Equation 2.

[0062] Since the component factors of the system posterior density depend only on subsets of system variables, the complexity of joint PDF estimation is reduced. Moreover, inference processing using the sum-product algorithm benefits from small factor size in terms of complexity and performance.

[0063] The disclosed technology, with respect to factor graphs, can be viewed as having three phases: 1. factor selection, 2. factor density estimation, and 3. execution of the sum-product algorithm with the learned model to make inferences based on new data. These phases are described in detail below.

[0064] Referring FIG. 1, there is shown a flow diagram of an operation for using supervised learning to create a factor graph without any predetermined graph structure and for applying the factor graph to new data to make inferences. Factor graphs are used as an example, and it will be understood that aspects of the present disclosure are applicable to probabilistic graphs in general.

[0065] At block 110, the operation involves accessing training data for variables of a data system to be modeled. The training data includes data for observable system variables. In various embodiments, the training data includes data for non-observable system variables.

[0066] At block 120, the operation involves processing the training data to identify subsets of system variables that are dependent. The operations of block 120 encompass the factor selection phase mentioned above. In this phase, subsets of variables can be tested for dependence by determining 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.

[0067] In accordance with aspects of the present disclosure, a technique to select factors follows from Corollary 1. In particular, the operation identifies dependent subsets of variables by checking how much their joint moments, measured from training data, deviate from the product of their individual moments. For example, if V_1 and V_2 represent dependent system variables, then there exists a pair of exponents (m, n) such that $0 < \Delta_{mn} \triangleq |E[V_1^m V_2^n] - E[V_1^m]E[V_2^n]|$. A larger deviation Δ_{mn} from zero indicates a greater degree of confidence that the variables exhibit dependence. Thus the Δ_{mn} serve as a metric for ranking dependent subsets. In the case of binary data, $E[V_1^m V_2^n] = E[V_1 V_2]$ for all m and n greater than zero. Hence, for binary data, it suffices to measure only the first order moments. In general, all orders of joint moments must be determined to fully characterize the joint probability density function of the subsystem of random variables, but there are exceptions, such as the binary case noted above and the Gaussian case which is fully characterized by its first and second order moments. However, heuristic criteria can be used to limit the search space, as described below.

[0068] In various embodiments, the training data may be 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, the operation can set: $\bar{v}_i[t] = \sigma_i^{-1}(v_i[t] - \mu_i)$, where $\mu_i = T^{-1}\sum v_i[t]$ and $\sigma_i^2 = T^{-1}\sum(v_i[t] - \mu_i)^2$. The operation then constructs the metrics $\Delta_{ij} = \left|T^{-1}\sum \bar{v}_i[t]\bar{v}_j[t]\right|$ and identifies those that exceed a predetermined threshold value. In various embodiments, the operation can add degree-three factors whose joint empirical moments exceed another predetermined threshold value. Hence, the operation can identify 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.

[0069] In various embodiments, due to the combinatorial complexity of factor detection for massive systems, heuristic criteria can be used to limit the search space for candidate variable subsets, such as those described above. In various embodiments, low-order/low-degree statistical moments (e.g., $m,n < 5$ or $m,n < 10$, etc.) can be used as the primary basis for selecting dependent subsets of variables. In various embodiments, approaches for reducing complexity include sparse graphs, clustering methods, random sampling, and physical models, among others.

[0070] At block 125, the operation involves creating a probabilistic graph, such as a factor graph, based on the subsets of variables that are identified as dependent. As mentioned above, the probabilistic graph that is created does not have any predetermined structure. Rather, in the case of factor graphs, the factor graph is created by creating a set of factor nodes corresponding to the dependent subsets of variables and by adding edges from the factors to their corresponding variable nodes.

[0071] Use of statistical moments to determine dependence and to create nodes and edges is merely an example, and heuristics or metrics other than statistical moments may be used. Such other heuristics or metrics may be used to create probabilistic graphs other than factor graphs. Such other heuristics and metrics and probabilistic graphs are contemplated to be within the scope of the present disclosure.

[0072] At block 130, the operation involves determining probabilistic parameters of the probabilistic graph created in block 125. More specifically, for a factor graph, the operation involves estimating a probability density function for each factor node of the factor graph using Monte Carlo integration. As persons skilled in the art will understand, Monte Carlo integration as

used herein refers to an empirical technique for determining the expected value of a statistical quantity which may be written as an integral of the quantity against its probability density function. The operations of block 130 encompass the factory density estimation phase mentioned above.

[0073] The operations of block 130 can characterize the functional form of the component factors of a global system PDF. The operation assumes that each component factor represents the joint PDF of its connected variables, i.e. posterior densities over the variables. For example, if $\{\boldsymbol{V_1}, \boldsymbol{V_2}\}$ and $\{\boldsymbol{V_2}, \boldsymbol{V_3}, \boldsymbol{V_4}\}$ were identified as the dependent subsets of variables of a four variable system, then the global PDF can be expressed by $\boldsymbol{f}(\boldsymbol{V_1}, \boldsymbol{V_2}, \boldsymbol{V_3}, \boldsymbol{V_4}) \propto \boldsymbol{f_1}(\boldsymbol{V_1}, \boldsymbol{V_2})\boldsymbol{f_2}(\boldsymbol{V_2}, \boldsymbol{V_3}, \boldsymbol{V_4})$. To solve for the functional form of the factor PDF, $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$, Theorem 1 can be used to construct estimates of the densities using empirical measurements of the joint statistical moments taken from training data via Monte Carlo integration as in Equation 2, or similarly, by directly computing the expected value of the complex exponential in Equation 2 via Monte Carlo integration. In various embodiments, the complexity of this phase may be mitigated by restricting attention to low-order moments or truncating the summations when building estimates of the component factors.

[0074] Monte Carlo integration is provided as an example for characterizing the functional form of factors and estimating probability density function from training data. Other embodiments are contemplated for characterizing the functional form of factors and estimating probability

density function from training data. Such other embodiments are contemplated to be within the scope of the present disclosure.

[0075] At block 135, the operation applies the probabilistic graph (e.g., factor graph) created at block 125 and the probabilistic graph's probabilistic parameters (e.g., component factor PDFs) determined at block 130 to new observed data to make a prediction.

[0076] Once the factor graph and its component densities are determined, the operation of block 135 implements an algorithm that applies the component densities to compute inferences based on new data. As an example, the algorithm may use an instance of the sum-product algorithm for computing inferences on certain system variables based on observed system variables, or using various message passing techniques. Aspects of implementing the sum-product algorithm are described in F. R. Kschischang, B. J. Frey and H. A. Loeliger, "Factor graphs and the sum-product algorithm," IEEE Transactions on Information Theory, vol. 47, no. 2, pp. 498-519, February 2001, which is hereby incorporated by reference herein in its entirety. Persons skilled in the art will understand how to implement the sum-product algorithm to compute inferences using component densities. Since multiple factor graphs can be used to model the same data system, it is advantageous to choose graphs with desirable properties for the inference algorithm. Sparse graphs with large girth may yield better performance. In various embodiments, low-degree factors may be selected before considering high-degree factors when constructing the factor graph model.

[0077] The descriptions provided in connection with FIG. 1 are examples. Variations are contemplated to be within the scope of the present disclosure.

[0078] Accordingly, aspects of the modeling and inference framework described above may be based on factor graphs and application of the sum-product algorithm for estimating the state of system variables to be predicted, given the state of observed system variables. Factor graphs

provide a basic framework for representing any data system as the product of several lower-complexity component densities. Given a specification of the component factors, the sum-product algorithm can be implemented directly from the graph representation to compute marginal densities on system variables to be predicted.

[0079] As an example of the operations of FIG. 1, the following will describe using automatic graph learning to reverse engineer the Hamming code, based on training data comprised of input-output codeword pairs. As shown below, 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.

[0080] The following describes 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.

[0081] The following also illustrates the application of factor graphs and the sum-product algorithm using a binary linear code, specifically the $(6,3)$ Hamming code, and describes applying 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, an abbreviated notation for the joint PDF may be used for compactness, where $f_{XY}(x,y)$ is written as $f(X,Y)$.

[0082] FIG. 2 depicts a factor graph relating the input and output bits of a $(6,3)$ Hamming code. The input bits u_i, $i = 1,2,3$ are collectively designated as 210 and the output bits c_i, $i = 1,2,\ldots,6$ are collective designated as 220. 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. The factors are collectively designated as 230.

[0083] Let $U = (U_1, \ldots, U_k)$ represent uncoded information bits 210 and $C = (C_1, \ldots, C_n)$ represent codewords 220 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, as defined in FIG. 3 for the $(6,3)$ Hamming code. The rows of H span the null space of G and correspond to factor nodes 230 of the factor graph representation of the code, as shown in FIG. 2. Any valid codeword must satisfy the parity check equations defined by the dual matrix.

[0084] Let $V = (U, C)$ denote the system variables. For the $(6,3)$ Hamming code, we can write a joint PDF for the system of variables as:

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

EQUATION 4

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 4 is expressed as a factor graph in FIG. 2. Whereas the information bits are usually omitted from the code's factor graph, these variable nodes are included to learn a model that is able to compute inferences on the variables to be predicted in a general system.

[0085] 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:

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

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:

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

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 220 in FIG. 2, reflecting that they are a function only of the $\{C_i\}$ given the channel output $\{Y_i\}$.

[0086] There is a one-to-one correspondence between the edges of the factor graph in FIG. 2 and the nonzero elements of the parity check matrix defined in FIG. 3. 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. Belief propagation comprises iterative message massing between nodes of the graph, as illustrated in FIG. 4 (variable node 210/220 update of belief propagation) and FIG. 5 (factor node 230 update of belief propagation). 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 appropriate 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.

[0087] Given a factor graph model, its component PDF's can be estimated using empirical measurements of relevant system moments derived from training data. In various embodiments, low-order, low-degree moments may be used to generate approximate estimates of factors of the system PDF. Given the graph model and estimates of its corresponding factors, the sum product

algorithm can be directly applied to compute the conditional PDF's of system variables of the graph to be predicted, given the status of observed system variables. Variable node processing and factor node processing steps of the sum-product algorithm are illustrated in FIG. 4 and FIG. 5, for the case of binary parity check data.

[0088] The following describes using the proposed AGL framework to reverse engineer the $(6,3)$ Hamming code based on training data of T input-output codeword pairs.

[0089] Phase I: Factor Selection

[0090] For the reverse engineering example, the thresholds are chosen to yield full rank parity check matrices with high reliability over randomly selected training data and used only degree-2 and degree-3 factors. In various embodiments, factor degrees can be selected according to a prescribed degree distribution derived from mutual information transfer analysis of the training data.

[0091] In various embodiments, other constraints can be imposed on the factor selection candidate set. For example, a linear independence condition can be enforced 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 prevents over-determining the factors of the system. Additionally, any new factor that overlaps with an existing factor by two or more variables can be discarded. 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 three iterations of belief propagation are guaranteed before the assumption of independence of factor messages is violated.

[0092] Phase 2: Factor Density Estimation

[0093] Equation 1 can be refined for the case of binary data:

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

EQUATION 5

where X and Y are binary random variables, and Equation 3 becomes:

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

EQUATION 6

Three or more variables can be generalized from the form above.

[0094] In the Error Control Coding (ECC) reverse engineering example described herein, Equation 6 is estimated from the training data with Monte Carlo integration and then Equation 5 is computed 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 various embodiments, the disclosed technology can jointly perform dependent subset selection and density estimation.

[0095] Phase 3: Sum Product Algorithm

[0096] The PDF computed with Equation 5 can be used to implement the sum-product algorithm as follows. When computing the factor node update for variable V_l, the conditional PDF $f(V_l|V_m,V_n)$ (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, for example, according to the equations defined in FIG. 4 and FIG. 5 which describe an implementation of sum-product update equations for binary data.

[0097] Accordingly, the phases described above in connection with FIGS. 2–5 create a factor graph with component factor densities that can operate as a decoder for a (6, 3) Hamming code. The factor graph is created without any predetermine graph structure. A comparison of factor graphs created using the present disclosure with various neural networks (FIG. 7) is described below.

[0098] FIG. 6 illustrates results of testing the performance of Automatic Graph Learning (AGL) versus Dense Neural Network (DNN) with different number of training codeword pairs (T=10,20,100). In the test results, AGL performs as well as belief propagation with a known decoder, both with 8 iterations.

[0099] The testing implemented a belief propagation decoder using an AGL derived model as described above and compared the performance to the true decoder in FIG. 6. 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.

[0100] 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. The AGL curve labeled "Model Mismatch Type 1" 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 "Model Mismatch Type 2" 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.

[0101] In 1000 trials of T=10 training codeword pairs, the testing 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 ("Model Mismatch Type 1"), and 2.3% had one or two orthogonality constraint violations ("Model Mismatch Type 2"). The numerical results further suggest that variations on the sum-product algorithm may compensate for model mismatch. In particular, the result labeled "Model Mismatch Type 2" was developed by a sum-product decoder that used the joint factor distribution when implementing check node processing in FIG. 4, instead of the conditional form as depicted. Variations on the inference processing algorithm, including variations on the sum-product algorithm and related algorithms, are included in the scope of the present invention.

[0102] The testing designed and trained multi-layer Dense Neural Networks (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 FIG. 7 (Keras model summary), where the Adam optimizer was used with a mean squared error loss function. The testing 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. Based on the test results, AGL has the potential to outperform competitive neural networks with an order of magnitude less training data.

[0103] Accordingly, the present disclosure provides an AGL framework for applications in machine learning and artificial intelligence systems. As described herein, a suitable factor graph model can be learned by detecting dependent subsets of system variables, and component factor PDFs 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.

[0104] In the disclosed approach, computational complexity may exist in constructing the factor graph, due to the combinatorial complexity of selecting variable subsets and of constructing an accurate model with limited training data. However, several heuristics are disclosed to reduce the candidate search space and, further, application specific prescriptions could be leveraged. For example, application specific constraints relating to statistical aspects of physical laws may be used to create the probabilistic model, which may help to reduce computational complexity. Once the graph is determined, the results show that a relatively small amount of training is usable to characterize the factor PDFs, such as only 10 training codewords to replicate the performance of the true belief propagation decoder. The results show that inference processing using belief propagation is viable even when there is a small degree of model mismatch.

[0105] Referring now to FIG. 8, there is shown a block diagram of exemplary components of a system or device 800. The block diagram is provided to illustrate possible implementations of various parts of the disclosed systems and devices, such as the operations of FIGS. 1–5.

[0106] The computing system 800 includes a processor or controller 805 that may be or include, for example, one or more central processing unit processor(s) (CPU), one or more Graphics Processing Unit(s) (GPU or GPGPU), and/or other types of processor, such as a microprocessor, digital signal processor, microcontroller, programmable logic device (PLD), field programmable gate array (FPGA), or any suitable computing or computational device. The computing system 800 also includes an operating system 815, a memory 820, a storage 830, input devices 835, output devices 840, and a communication device 822. The communication device 822 may include one or more transceivers which allow communications with remote or external

devices and may implement communications standards and protocols, such as cellular communications (e.g., 3G, 4G, 5G, CDMA, GSM), Ethernet, Wi-Fi, Bluetooth, low energy Bluetooth, Zigbee, Internet-of-Things protocols (such as mosquitto MQTT), and/or USB, among others.

[0107] The operating system 815 may be or may include any code designed and/or configured to perform tasks involving coordination, scheduling, arbitration, supervising, controlling or otherwise managing operation of computing system 800, such as scheduling execution of programs. The memory 820 may be or may include, for example, one or more Random Access Memory (RAM), read-only memory (ROM), flash memory, volatile memory, non-volatile memory, cache memory, and/or other memory devices. The memory 820 may store, for example, executable instructions that carry out an operation (e.g., executable code 825) and/or data. Executable code 825 may be any executable code, e.g., an app/application, a program, a process, task or script. Executable code 825 may be executed by controller 805.

[0108] The storage 830 may be or may include, for example, one or more of a hard disk drive, a solid state drive, an optical disc drive (such as DVD or Blu-Ray), a USB drive or other removable storage device, and/or other types of storage devices. Data such as instructions, code, procedure data, and medical images, among other things, may be stored in storage 830 and may be loaded from storage 830 into memory 820 where it may be processed by controller 805. The input devices 535 may include, for example, a mouse, a keyboard, a touch screen or pad, or another type of input device. The output devices 840 may include one or more monitors, screens, displays, speakers and/or other types of output devices.

[0109] The illustrated components of FIG. 8 are exemplary and variations are contemplated to be within the scope of the present disclosure. For example, the numbers of components may be

greater or fewer than as described and the types of components may be different than as described. When the system 800 implements a machine learning system, for example, a large number of graphics processing units may be utilized. When the computing system 800 implements a data storage system, a large number of storages may be utilized. As another example, when the computing system 800 implements a server system, a large number of central processing units or cores may be utilized. Other variations and applications are contemplated to be within the scope of the present disclosure.

[0110] The embodiments disclosed herein are examples of the disclosure and may be embodied in various forms. For instance, although certain embodiments herein are described as separate embodiments, each of the embodiments herein may be combined with one or more of the other embodiments herein. Specific structural and functional details disclosed herein are not to be interpreted as limiting, but as a basis for the claims and as a representative basis for teaching one skilled in the art to variously employ the present disclosure in virtually any appropriately detailed structure. Like reference numerals may refer to similar or identical elements throughout the description of the figures.

[0111] The phrases "in an embodiment," "in embodiments," "in various embodiments," "in some embodiments," or "in other embodiments" may each refer to one or more of the same or different embodiments in accordance with the present disclosure. A phrase in the form "A or B" means "(A), (B), or (A and B)." A phrase in the form "at least one of A, B, or C" means "(A); (B); (C); (A and B); (A and C); (B and C); or (A, B, and C)."

[0112] The systems described herein may also utilize one or more processors to receive various information and transform the received information to generate an output. The processor may include any type of computing device, computational circuit, or any type of controller or

processing circuit capable of executing a series of instructions that are stored in a memory. The processor may include multiple processors and/or multicore central processing units (CPUs) and/or graphical processing units (GPUs) and may include any type of controller, such as a microprocessor, digital signal processor, microcontroller, programmable logic device (PLD), field programmable gate array (FPGA), or the like. The processor may also include a memory to store data and/or instructions that, when executed by the one or more processors, causes the one or more processors to perform one or more methods and/or algorithms.

[0113] Any of the herein described methods or operations may be converted to, or expressed in, a programming language or computer program. The terms "programming language" and "computer program," as used herein, each include any language used to specify instructions to a computer, and include (but is not limited to) the following languages and their derivatives: Assembler, Basic, Batch files, BCPL, C, C+, C++, Delphi, Fortran, Java, JavaScript, machine code, operating system command languages, Pascal, Perl, PL1, Python, scripting languages, Visual Basic, metalanguages which themselves specify programs, and all first, second, third, fourth, fifth, or further generation computer languages. Also included are database and other data schemas, and any other meta-languages. No distinction is made between languages which are interpreted, compiled, or use both compiled and interpreted approaches. No distinction is made between compiled and source versions of a program. Thus, reference to a program, where the programming language could exist in more than one state (such as source, compiled, object, or linked) is a reference to any and all such states. Reference to a program may encompass the actual instructions and/or the intent of those instructions.

[0114] It should be understood that the foregoing description is only illustrative of the present disclosure. Various alternatives and modifications can be devised by those skilled in the art without

departing from the disclosure. Accordingly, the present disclosure is intended to embrace all such alternatives, modifications, and variances. The embodiments described with reference to the attached drawing figures are presented only to demonstrate certain examples of the disclosure. Other elements, steps, methods, and techniques that are insubstantially different from those described above and/or in the appended claims are also intended to be within the scope of the disclosure.

What is Claimed:

1. A system comprising:

one or more processors; and

at least one memory storing instructions which, when executed by the one or more processors, cause the system to:

access training data relating to a plurality of variables,

determine a factor graph model based on the training data, the factor graph model including component factors,

estimate probability density functions for the component factors based on Monte Carlo integration in frequency domain, and

apply the factor graph model with the estimated probability density functions for the component factors to new observed data to provide a prediction.

2. The system of claim 1, wherein in determining the factor graph model, the instructions, when executed by the one or more processors, cause the system to determine low-order moments based on the training data.

3. The system of claim 2, wherein in determining the factor graph model, the instructions, when executed by the one or more processors, cause the system to identify subsets of dependent variables based on the low-order moments.

4. The system of claim 3, wherein in identifying the subsets of dependent variables, the instructions, when executed by the one or more processors, cause the system to compare whether joint variable moments deviate from products of individual variable moments.

5. The system of claim 4, wherein in determining the factor graph model, the instructions, when executed by the one or more processors, cause the system to create factor nodes corresponding to the subsets of dependent variables and adding edges from the factor nodes to corresponding variable nodes.

6. The system of claim 1, wherein in estimating probability density functions for the component factors, the instructions, when executed by the one or more processors, cause the system to estimate functional forms of the probability density functions for the component factors using Fourier domain synthesis.

7. The system of claim 6, wherein in estimating probability density functions for the component factors, the instructions, when executed by the one or more processors, cause the system to estimate the probability density functions for the component factors using the functional forms, joint variable moments, and direct Monte Carlo integration in the frequency domain.

8. The system of claim 1, wherein the instructions, when executed by the one or more processors, further cause the system to apply the factor graph model with the estimated

probability density functions for the component factors to observed variables to make an inference.

9. A method comprising:

accessing training data relating to a plurality of variables;

determining a factor graph model based on the training data, the factor graph model including component factors;

estimating probability density functions for the component factors based on Monte Carlo integration in frequency domain; and

applying the factor graph model with the estimated probability density functions for the component factors to new observed data to make a prediction.

10. The method of claim 9, wherein determining the factor graph model includes determining low-order moments based on the training data.

11. The method of claim 10, wherein determining the factor graph model includes identifying subsets of dependent variables based on the low-order moments.

12. The method of claim 11, wherein identifying the subsets of dependent variables includes comparing whether joint variable moments deviate from products of individual variable moments.

13. The method of claim 12, wherein determining the factor graph model includes creating factor nodes corresponding to the subsets of dependent variables and adding edges from the factor nodes to corresponding variable nodes.

14. The method of claim 9, wherein estimating the probability density functions for the component factors includes estimating functional forms of the probability density functions for the component factors using Fourier domain synthesis.

15. The method of claim 14, wherein estimating the probability density functions for the component factors includes estimating the probability density functions for the component factors using the functional forms, joint variable moments, and direct Monte Carlo integration in the frequency domain.

16. The method of claim 9, further comprising applying the factor graph model with the estimated probability density functions for the component factors to observed variables to make an inference.

17. A method comprising:

accessing training data relating to a plurality of variables;

determining a probabilistic graph model based on the training data, the probabilistic graph model including nodes and edges deduced from the training data, and the probabilistic graph model having no predetermined structure prior to the nodes and edges being deduced;

estimating probabilistic parameters of the probabilistic graph model using the training data; and

applying the probabilistic graph model with the estimated probabilistic parameters to new observed data to make a prediction.

18. The method of claim 17, wherein the probabilistic graph model is a factor graph model.

ABSTRACT

The present disclosure relates to supervised graph learning. In aspects, a system includes one or more processors and at least one memory storing instructions. The instructions, when executed by the processor(s), cause the system to access training data relating to variables, determine a factor graph model based on the training data where the factor graph model includes component factors, estimate probability density functions for the component factors based on Monte Carlo integration in the frequency domain, and apply the factor graph model with the estimated probability density functions for the component factors to new observed data to make a prediction.

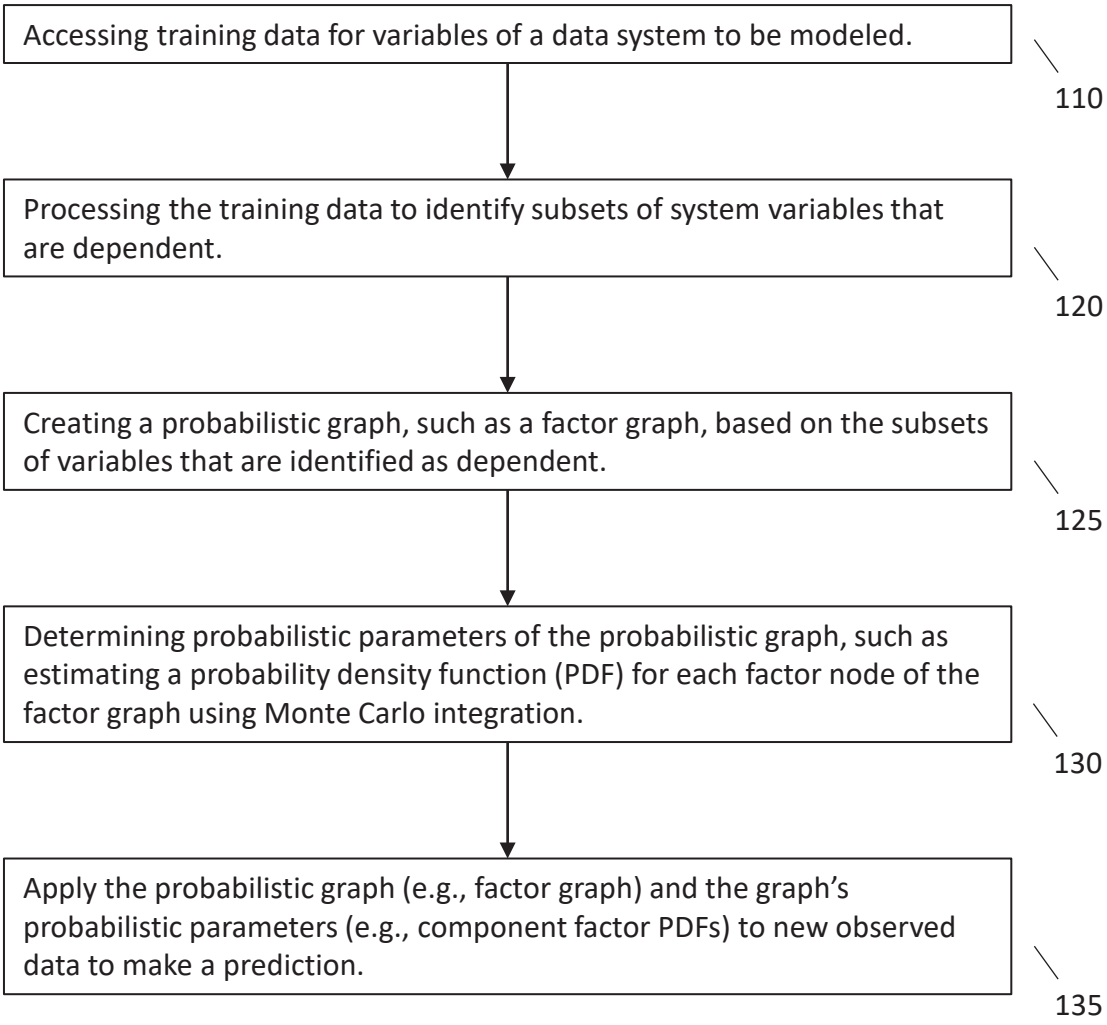
Accessing training data for variables of a data system to be modeled.
110
Processing the training data to identify subsets of system variables that are dependent.
120
Creating a probabilistic graph, such as a factor graph, based on the subsets of variables that are identified as dependent.
125
Determining probabilistic parameters of the probabilistic graph, such as estimating a probability density function (PDF) for each factor node of the factor graph using Monte Carlo integration.
130
Apply the probabilistic graph (e.g., factor graph) and the graph's probabilistic parameters (e.g., component factor PDFs) to new observed data to make a prediction.
135

FIG. 1

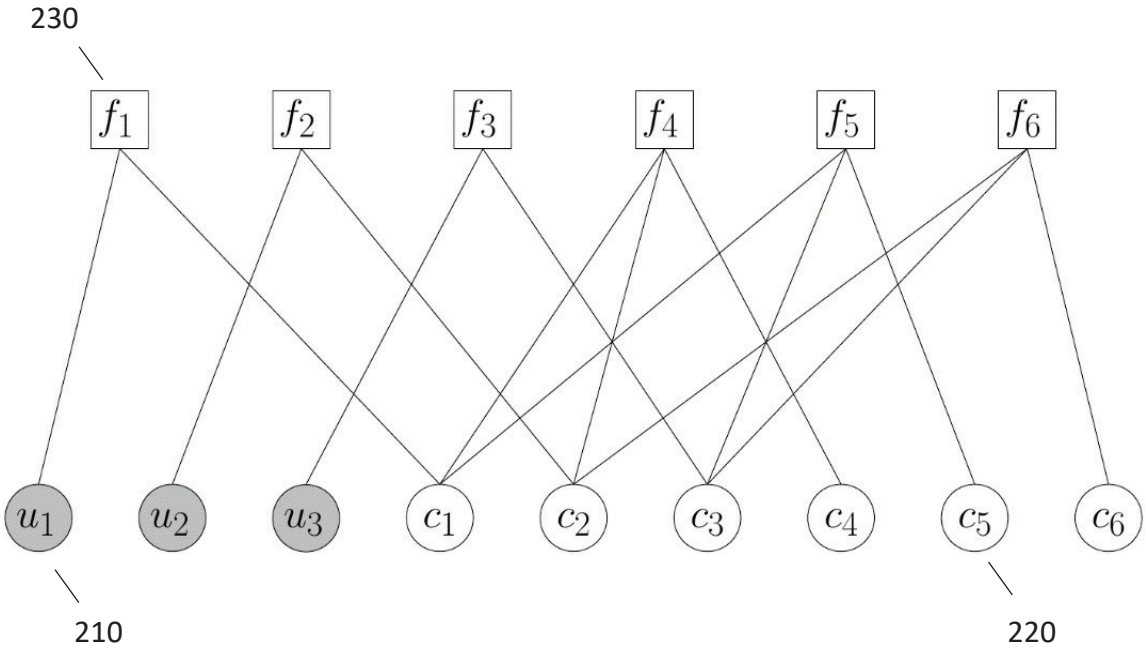

FIG. 2

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. 3

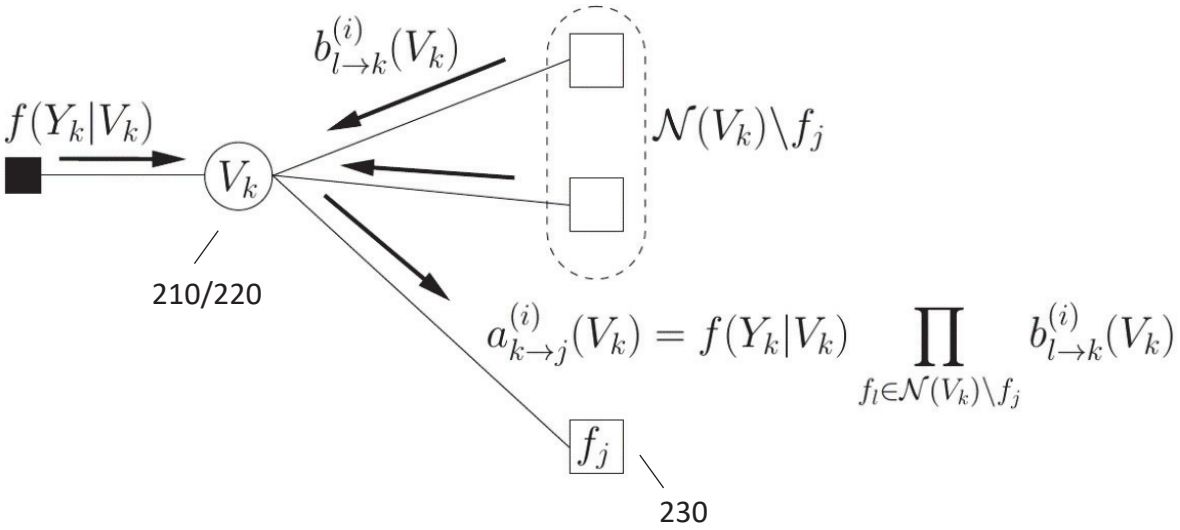
$b_{l\to k}^{(i)}(V_k)$
$f(Y_k|V_k)$
V_k
$\mathcal{N}(V_k)\backslash f_j$
210/220
$a_{k\to j}^{(i)}(V_k) = f(Y_k|V_k) \prod_{f_l\in\mathcal{N}(V_k)\backslash f_j} b_{l\to k}^{(i)}(V_k)$
f_j
230

FIG. 4

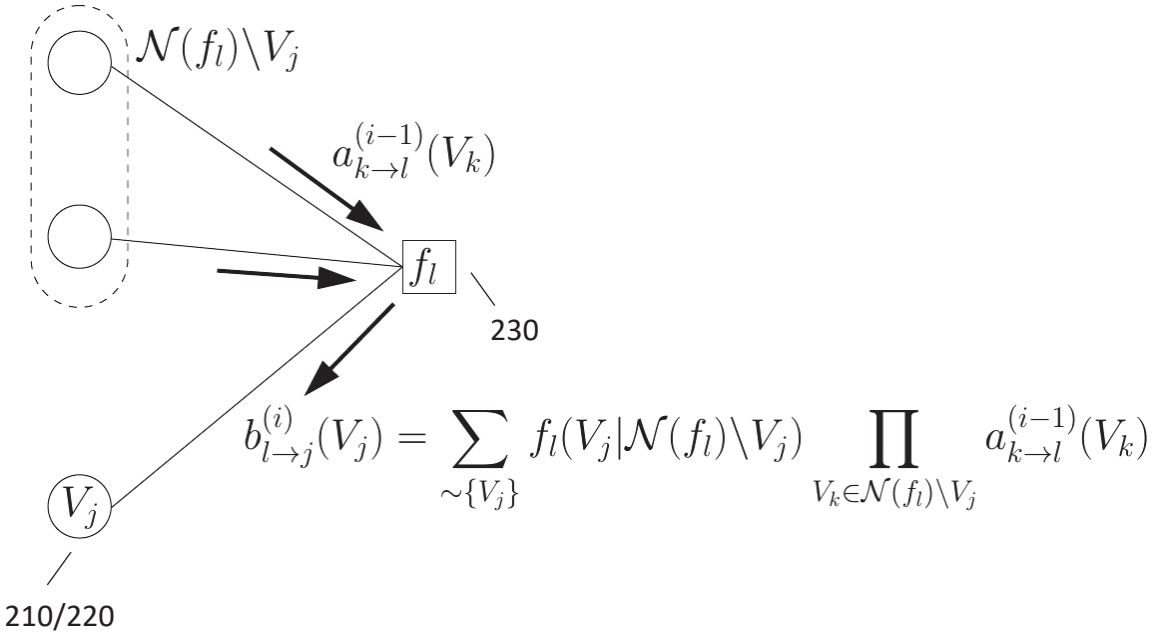
$\mathcal{N}(f_l)\backslash V_j$
$a_{k\to l}^{(i-1)}(V_k)$
f_l
230
$b_{l\to j}^{(i)}(V_j) = \sum_{\sim\{V_j\}} f_l(V_j|\mathcal{N}(f_l)\backslash V_j) \prod_{V_k\in\mathcal{N}(f_l)\backslash V_j} a_{k\to l}^{(i-1)}(V_k)$
V_j
210/220

FIG. 5

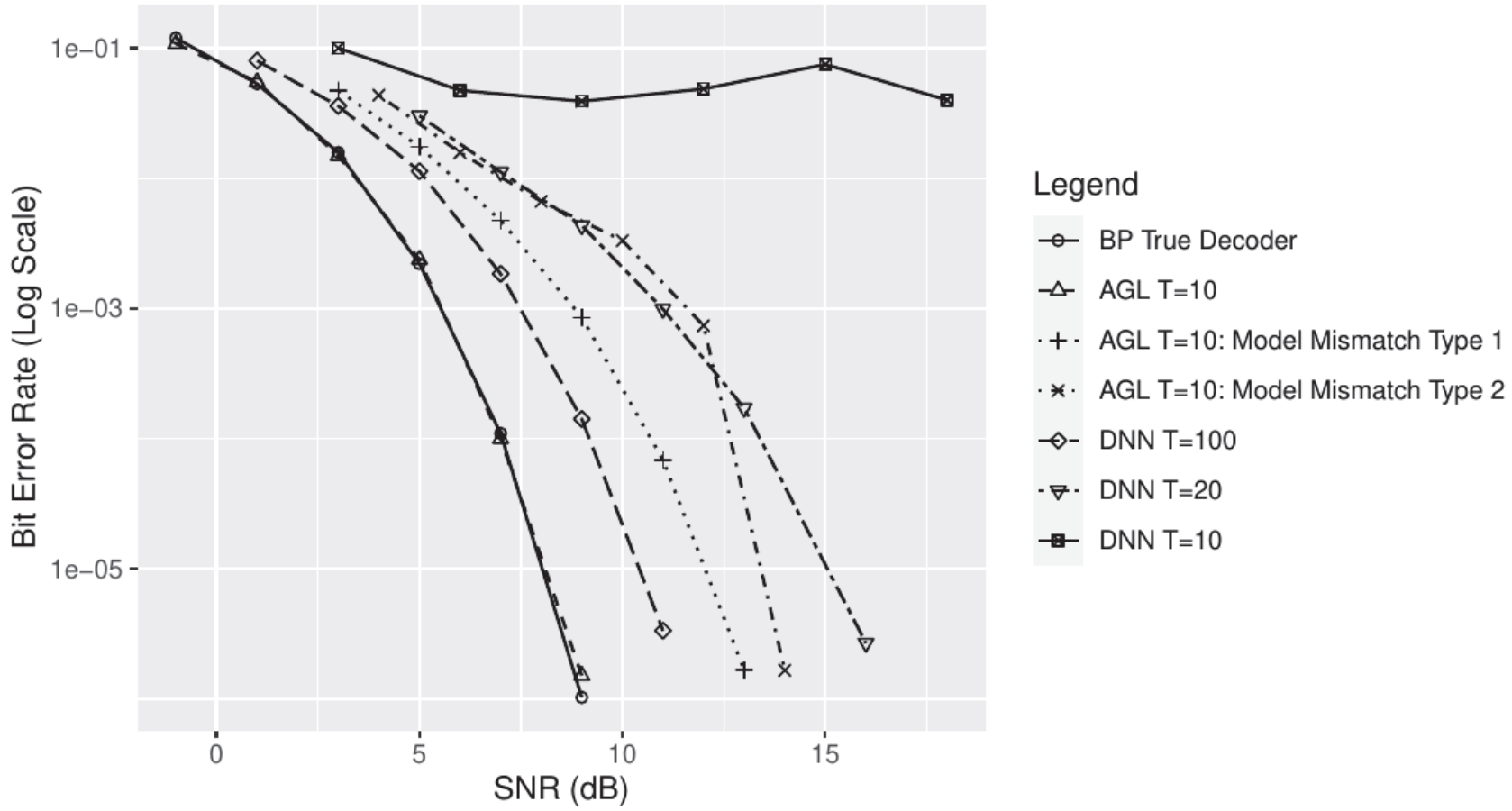
Bit Error Rate (Log Scale)
1e−01
1e−03
1e−05
0
5
10
15
SNR (dB)
Legend
BP True Decoder
AGL T=10
AGL T=10: Model Mismatch Type 1
AGL T=10: Model Mismatch Type 2
DNN T=100
DNN T=20
DNN T=10

FIG. 6

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. 7

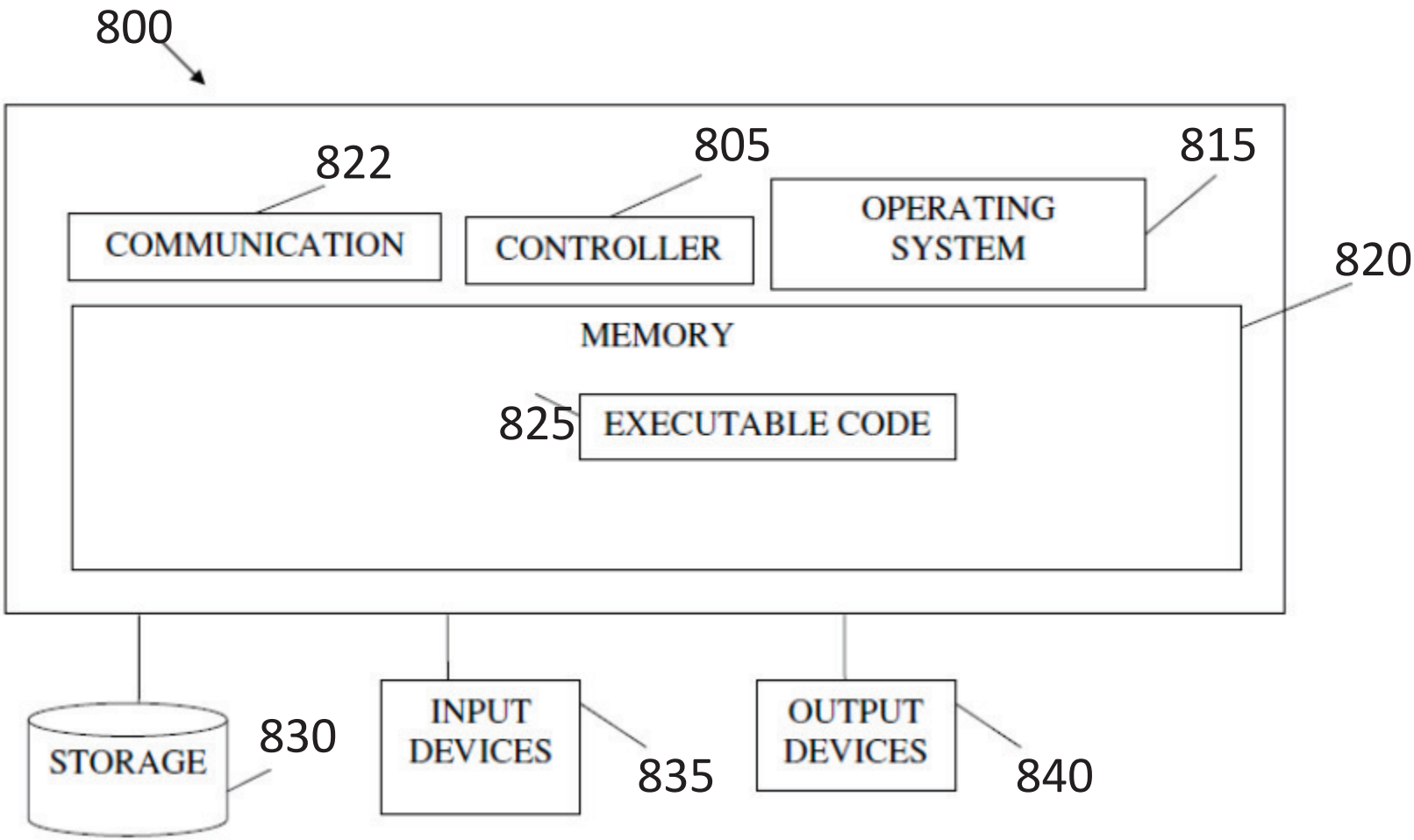
800
822
805
815
COMMUNICATION
CONTROLLER
OPERATING SYSTEM
820
MEMORY
825
EXECUTABLE CODE
STORAGE
830
INPUT DEVICES
835
OUTPUT DEVICES
840

FIG. 8