Knowledge-Based Systems 108 (2016) 144–154

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Learning word dependencies in text by means of a deep recurrent belief network

Iti Chaturvedi[a], Yew-Soon Ong[a], Ivor W. Tsang[b], Roy E. Welsch[c], Erik Cambria[a,*]

[a] School of Computer Science and Engineering, Nanyang Technological University, Singapore
[b] Center for Quantum Computation and Intelligent Systems, University of Technology, Sydney, Australia
[c] Sloan School of Management, MIT, USA

ARTICLE INFO

Article history:
Received 16 November 2015
Revised 8 July 2016
Accepted 12 July 2016
Available online 13 July 2016

Keywords:
Deep belief networks
Time-delays
Variable-order
Gaussian networks
Markov Chain Monte Carlo

ABSTRACT

We propose a deep recurrent belief network with distributed time delays for learning multivariate Gaussians. Learning long time delays in deep belief networks is difficult due to the problem of vanishing or exploding gradients with increase in delay. To mitigate this problem and improve the transparency of learning time-delays, we introduce the use of Gaussian networks with time-delays to initialize the weights of each hidden neuron. From our knowledge of time delays, it is possible to learn the long delays from short delays in a hierarchical manner. In contrast to previous works, here dynamic Gaussian Bayesian networks over training samples are evolved using Markov Chain Monte Carlo to determine the initial weights of each hidden layer of neurons. In this way, the time-delayed network motifs of increasing Markov order across layers can be modeled hierarchically using a deep model. To validate the proposed Variable-order Belief Network (VBN) framework, it is applied for modeling word dependencies in text. To explore the generality of VBN, it is further considered for a real-world scenario where the dynamic movements of basketball players are modeled. Experimental results obtained showed that the proposed VBN could achieve over 30% improvement in accuracy on real-world scenarios compared to the state-of-the-art baselines.

1. Introduction

Dynamic Gaussian networks (GN) have shown success in capturing temporal characteristics of data in sports and text processing [1]. They assume an underlying hidden state of a dynamic system evolving over time. For example, in basketball, the structure of the GN can be learned from the temporal movement of players over time and determine the differences in offensive formations between expert and beginners. Hence, the directed edges represent causal dependencies among the players and the time-delays associated with the edges define the dynamics. Dynamic GN are stochastic models where learning involves enumerating different local connectivity patterns consisting of child nodes given parent nodes at the previous two or three time points that may re-occur in one or more classes [2]. Such a directed sub-graph is referred to as a 'network motif'.

Although, GN outperform state-of-the-art classifiers including Bayesian Networks (BN) and Differential equations [3], training them requires large number of time samples. Hence, in this paper we consider a framework similar to feed-forward neural networks and extreme learning machines that can automatically learn temporal features with long term memory in a fast and easy manner with minimal human intervention and limited time samples [4,5].

Further, GN need to use additional memory nodes to learn time delays. Hence, the total number of nodes and motifs increases exponentially at very long delays, making it computationally intractable. Conditional random fields alleviate this problem by approximating a very long delay as a cascade of short delays to summarize text [6]. However, in long documents or reviews this can lead to formation of long and overlapping loops. Our goal is to efficiently predict the next class label in a sequence for high-dimensional networks and this can be done using the hierarchical structure of deep neural networks.

Recently, recurrent neural networks (RNN) have exhibited good performance for modeling temporal structures with few training samples [7]. This is because bi-directional RNN can model not only

Abbreviations: *BN* =, Bayesian Network; *CD* =, Contrastive Divergence; *DBN* =, Deep Belief Network; *GN* =, Gaussian Network; *RNN* =, Recurrent Neural Network; *VBN* =, Variable-order Belief Network; *MCMC* =, Markov Chain Monte Carlo; *ML* =, Maximum Likelihood; *MVAR* =, Multivariate Autoregression; *RBM* =, Restricted Boltzmann Machine.

* Corresponding author.
E-mail address: cambria@ntu.edu.sg (E. Cambria).

http://dx.doi.org/10.1016/j.knosys.2016.07.019
0950-7051/

Notations

$x_i(\tau)$	Expression level of node i at time instant τ
$\boldsymbol{\theta}_i$	Parameters for node i in the Bayesian network
$\boldsymbol{a}_i$	Parent set of a node i
N	Number of variables in the system
v_i	Node i in the visible layer
h_j	Node j in the hidden layer
T	Number of data samples
r	index for order of delay
R	The upper-bound of delay
n_l	The number of nodes in the layer l
l	Index for a hidden layer
L	Total number of layers
f	Activation function of each hidden neuron
$g(x(\tau))$	Joint probability over all nodes at time instant τ
W_l	Weights of the hidden layer l
W_r	Weights of r-order edges among visible nodes
α	Learning rate of a DBN
S	Gaussian network

the past but also the future that is useful in sentence completion tasks [8]. However, RNN require additional memory neurons to model each time delay and training becomes difficult as the gradient declines sharply with increasing delay. This can also result in unstable convergence, as the Hessian matrix of second-order derivatives does not exist for many real datasets.

The main difference from the work done in [9] is that instead of resorting to Hessian free optimization to learn RNN with time-delays, we take cue from the fact that RNN are deep neural networks with weight sharing across time. Hence, we consider deep learning where hierarchies of modules can provide a compact representation to temporal features in the form of input-output pairs. Contrary to previous approaches, we propose a variable-order deep Belief Network (VBN) that uses a dynamic Gaussian Bayesian network and Markov Chain Monte Carlo (MCMC) sampling to reduce the dimensionality of temporal problems without any loss of information. This is achieved from our knowledge of time delays; we can learn short delays independent of long delays in a hierarchical manner, since the former is a part of the latter. Here, we train each additional hidden layer of neurons with recurrent motifs extracted from the time series data of increasing Markov order using dynamic Gaussian Bayesian networks.

In order to reduce the complexity of the model each hidden layer is a restricted Boltzmann machine (RBM) that is learned independent of the others. In particular, to train the proposed VBN, we extract time-delayed features in the form of dynamic network motifs from the original time series using MCMC. Fig. 1(a) illustrates a deep belief network where the input nodes are a connectivity matrix of width N; the maximum order of delay is R and there are L RBM layers. Given input vector **x**, each hidden neuron in the lth RBM layer is designed to learn weights W_{ij}^{l}. Fig. 1(b) illustrates l-order hidden neurons that learns from the inputs with up-to l-order time delays to arrive at the weights W_j^l of hidden neuron j. In contrast to previous methods of duplicating neurons to model time-delays, here dynamic Gaussian Bayesian networks over training motifs are evolved using Markov Chain Monte Carlo to perturb the initial weights of each hidden layer of neurons to include time-delays. VBN does not require any additional memory neurons as delays are cascaded. In this way they do not need to pass through the non-linearity at each time point and the loss in gradient is much lower than in the case of RNN.

For instance, the number of possible first-order time delayed features is exponential; hence if we initialize the weights of the hidden neurons of first RBM layer using high probability first-order motifs, it is then much easier to train using contrastive divergence. Similarly, for the second RBM layer we initialize the weights using high probability second-order motifs. Since, both the first and second-order network motifs are learned from the same training data, they will belong to the same distribution.

Further, as explained in [10,11], in such a layered model, the features or network motifs learned in the first layer become input to the second layer and so on. For example, the first RBM layer will learn network motif representations with only first-order time delays by minimizing the error between each initial motif and the corresponding motif predicted by the hidden neurons in the training data. Next, the training motifs are evolved to include second-order time delayed edges prior to training the second RBM layer. The first RBM layer will now try to emit each second-order time delayed edge as a cascade of two first-order time delayed edges occurring in different features that will become the input to the second RBM layer.

In such a hierarchy of predictors, the input at any given time at one level is coming from the previous level. Hence, it is sufficient to know those elements of the input data that were not correctly predicted. The error function of each module forces it to emit a learned target representation in the input data. If the module makes an error, the unpredicted input will be transformed to a unique representation and send to the next higher module. To our knowledge, such a framework that can combine small delays to model long delays via deep MCMC sampling has previously not been proposed.

2. Related work and contributions

Depending on the problem, learning in a neural network may require long causal chains of computational stages. To reduce the redundancy in the data and consequently depth of the model, unsupervised learning methods such as Boltzmann machines are used that maximize the entropy related information [12]. Unsupervised learning can automatically generate sparse representations of inputs using well-known feature detectors such as edge detectors or Gabor filters. From then on, only unexpected inputs (errors) convey new information and are fed to the next higher layer. For each individual input sequence, we get a series of less and less redundant encoding in deeper and deeper levels also known as history compression. In convolution NN, a filter of shared weights is shifted step by step over a 2D array of inputs resulting in massive weight sharing. Each convolution layer is inter-leaved with a max-pooling layer [13]. In max pooling, each convolution layer is replaced by a down sampling layer by the activation of its maximally active unit. By eliminating non-maximal values, it can reduce redundancies due to convolution.

Finally, Deep Belief Network (DBN) is a stack of Restricted Boltzmann Machines [14]. Each RBM takes as input the pattern representations from the level below and learns to encode them in an unsupervised fashion. Occam's razor suggests that a NN with low weight complexity corresponds to high NN accuracy without over fitting to training data. Each RBM layer results in a reduction in dimensionality of the input as long as the number of hidden neurons is lower than previous layer. Hence, the corresponding minimum description length of the data or the negative log probability of the data will keep improving [15]. Lastly, the DBN can be fine-tuned using back-propagation.

Computing the probability of an RBM is difficult, as the normalization constant requires summation over all possible configurations of the hidden neurons. In [16] the authors proposed a heuristic approach called Contrastive Divergence(CD) that tries to minimize the Kullback–Leibler divergence between the input samples and target distribution. Here, we use Gibbs sampling over each

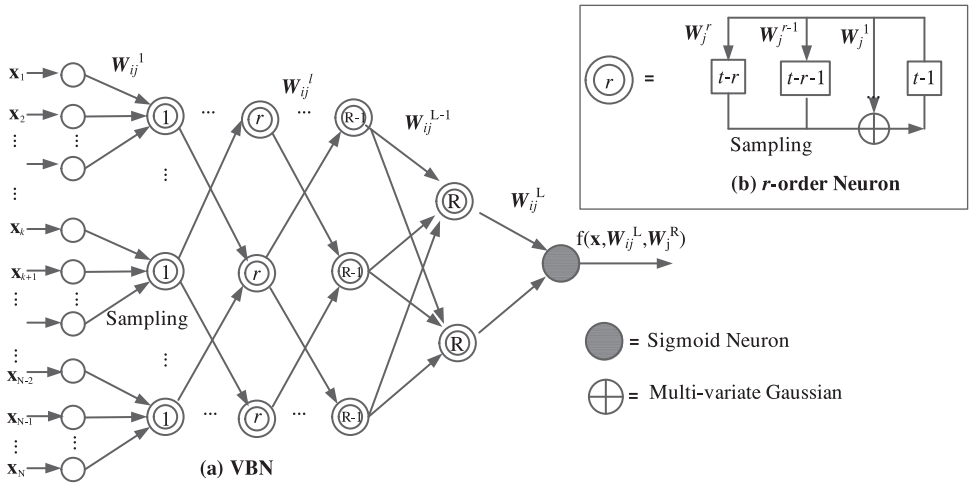

Fig. 1. (a) Illustrates a VBN with N input nodes (b) Illustrates neurons that learns from inputs with up-to *r*-order time delays.

input sample for a small number of *k* steps. In each step, the hidden neurons are sampled from a multivariate distribution over visible nodes, and then the visible nodes are reconstructed as samples from a multivariate distribution over the hidden neurons. Next, we update the weights to minimize the error between the input samples and the reconstructed visible samples in the last step. Low-level features learned in one layer are combined to form complex high-level features in the next higher layer. In speech recognition for example, deep belief networks have been reported to offer improved accuracy over state-of-the-art approaches in estimating the transition probabilities of a hidden Markov model [17,18]. Similarly, for video classification, deep belief networks have provided a scalable translation-invariant option by dividing frames into blocks learned via convolution [19].

Much work has been done on learning static representations using feed-forward NN where the connections among the neurons cannot form loops; These have wide application in classification of text where context in a sequence of words can be captured using higher-order dependencies [20–22]. The earliest time-delayed neural networks convolved the input with respect to time to model delays using duplicate hidden neurons. In such a model, number of neurons increases exponentially at long delays, making such models computationally intractable [14].

Due to the absence of connections among neurons in the same layer, the RBM can only learn a first-order Markov sequence from the data and is unable to represent higher-order time delays or loops in the form of feedbacks. In this paper, we resort to dynamic Gaussian Bayesian networks to extract network motifs with higher-order time delays. The hidden neurons are initialized with high ML network motifs. For each new layer we increase the upper bound on time delays in a network motif used for pre-training the neuron weights of that layer. This is implemented by interleaving a variable-order Markov Chain Monte Carlo (VMCMC) with dynamic Gaussian Bayesian fitness function between the visible nodes and the first layer of hidden neurons [23]. The resulting layer is referred to as a variable-order Boltzmann machine (VBM). Here, we refer to the proposed resultant framework as variable-order DBN (VBN), where each layer is now a VBM, in place of the original RBM.

The significance and contributions of the research work presented in this paper can be summarized as follows:

- We introduce a new variable-order Boltzmann machine (VBM) capable of modeling dynamic systems. From our knowledge, no previous work has considered efficient learning of Gaussian networks with long time delays using deep belief networks.
- A deep stochastic search algorithm is used to aid the learning of time delays in the Gaussian networks. Our results show that the method is robust even when few training samples are available.

Validation of the method is then performed using benchmark real-world datasets. Comparison is done with baseline structural learning method namely Bayesian Networks (BN) [24] as well as classifiers such as Conditional Random Fields (CRF), Single layer feed-forward neural network (SLFN) and Recurrent Neural Network(RNN).

First, we applied the VBN on real-world data collected from basketball matches to demonstrate the generality of the proposed approach. Recently, many authors have applied game theory to model complex system dynamics. This is because in team sports such as basketball, the trajectories of players change rapidly resulting in different offensive formations. Hence, we consider this as a good practical example where long-term memory is used. In particular, the trajectories of five players in basketball matches are considered for classifying a match as either a beginner game or a skilled game. To achieve this, we used the trajectory data to determine the maximum likelihood estimates of the Gaussian network, corresponding to the offensive formations observed in the game. We consider offensive formations in four classes 'Beginner Winner', 'Beginner Loser', 'Skilled Winner', and 'Skilled Loser'. The VBN was able to classify the skilled games with much higher accuracy compared to baselines.

Next, to further verify the effectiveness of VBN in capturing the dependencies in high-dimensional data, we consider here the processed version of the "20 Newsgroups" dataset, which has over 12,000 documents from 20 groups. Each document is featured by a sequence of words with a large vocabulary and variable-order dependencies in a sentence. The classification accuracy for several groups obtained using the proposed VBN is shown to significantly outperform the baseline state-of-the-art Stanford classifier which employed conditional random fields [25].

Nevertheless, the proposed framework can also be used in different high-dimensional temporal problems such as video classification or stock market prediction. We did not show benchmarks on video classification as they are highly similar to video tracking of basketball players.

Another useful application is sentiment analysis [26], where several neutral sentences may occur in between positive or negative comments in product reviews [27–29]. We also did not show benchmarks on sentiment analysis as it is a binary classification problem much easier to the classification of documents or sentences into 20 different topics.

The rest of the paper is organized as follows: Section 3 provides the preliminary concepts necessary to comprehend the proposed VBN algorithm of the present work. In Section 4, we introduce the proposed variable-order Boltzmann machine and describe the algorithm for learning the weights of a VBN framework. Lastly, in Section 5, we validate our method on real-world benchmark datasets.

3. Preliminaries

In this section, we briefly review the theoretical concepts necessary to comprehend the present work. We begin with a description of multivariate regression models. This is followed by a description of maximum likelihood estimation of edges in the structure using the stochastic search called Markov Chain Monte Carlo with dynamic Gaussian Bayesian networks as a fitness function. Next, we show that weights in the deep belief network can be learned by maximizing a global energy function that corresponds to an exponential distribution over a linear combination of input features such as the high ML edges of a network.

Notations: Consider a Gaussian network (GN) with time delays which comprises a set of N nodes and observations gathered over T instances for all the nodes. Nodes can take real values from a multivariate distribution determined by the parent set. Let the dataset of samples be $X = \{x_i(\tau)\}_{\mathrm{N}\times\mathrm{T}}$, where $x_i(\tau)$ represents the sample value of the i^{th} random variable in instance τ. Lastly, let $\boldsymbol{a}_i$ be the set of parent variables regulating variable i.

3.1. Multivariate autoregression

A multivariate autoregressive (MVAR) model uses differential equations to predict the GN with time delays involving a set of random variables [30]. We denote the upper bound of delay in the network with R. An R-order MVAR model represents linear regression with delays:

$$x(\tau) = \sum_{r=1}^{\mathrm{R}} \beta^t \mathbf{x}(\tau - r) + \boldsymbol{\varepsilon}(\tau), \tag{1}$$

where $\beta^\tau = \{\beta^\tau_{i,j}\}_{\mathrm{N}\times\mathrm{N}}$ denotes the matrix of regression coefficients corresponding to a delay of τ time points, and $\boldsymbol{\varepsilon}(\tau) = (\varepsilon_i(\tau))_{i=1}^{\mathrm{N}}$ denotes i.i.d. zero mean additive Gaussian noise. The regression coefficient $\beta^\tau_{i,j}$ when $i \neq j$, represents the dependence between the two nodes: i and j [31]. The edges of the network are obtained via forcing normalized coefficients less than 0.5 to 0. However, with increasing size and cross talk in the network, it is more efficient to represent GN as a graphical model such as the Bayesian network (BN).

3.2. Dynamic Gaussian Bayesian networks

A Bayesian network is a graphical model that represents a joint multivariate probability distribution for a set of random variables [30]. It is a directed acyclic graph S with a set of parameters $\boldsymbol{\theta}$ that represents the strengths of connections by conditional probabilities. The BN decomposes the likelihood of node expressions into a product of conditional probabilities by assuming independence of non-descendant nodes, given their parents.

$$p(X|S,\boldsymbol{\theta}) = \prod_{i=1}^{\mathrm{N}} p(\mathbf{x}_i|\boldsymbol{a}_i, \theta_{i,\boldsymbol{a}_i}), \tag{2}$$

where $p(\mathbf{x}_i|\boldsymbol{a}_i, \theta_{i,\boldsymbol{a}_i})$ denotes the conditional probability of node expression $\mathbf{x}_i$ given its parent node expressions $\boldsymbol{a}_i$, and $\theta_{i,\boldsymbol{a}_i}$ denotes the ML estimate of the conditional probabilities. In a multivariate dynamic graph, each node is a variable in the state-space of the system that can be observed or measured. The connections represent causal dependencies over one or more time instants The observed state vector of variable i is denoted as $\mathbf{x}_i$ and the conditional probability of variable i given variable j is $p(\mathbf{x}_i|\mathbf{x}_j)$.

The optimal Gaussian network S^* is obtained by maximizing the posterior probability of S given the data X. From Bayes theorem, the optimal Gaussian network S^* is given by

$$S^* = \arg\max_S p(S|X) = \arg\max_S p(S)p(X|S), \tag{3}$$

where $p(S)$ is the probability of the Gaussian network and $p(X|S)$ is the likelihood of the expression data given the Gaussian network.

Given the set of conditional distributions with parameters $\boldsymbol{\theta} = \{\theta_{i,\boldsymbol{a}_i}\}_{i=1}^{\mathrm{N}}$, the likelihood of the data is given by

$$p(X|S) = \int p(X|S,\boldsymbol{\theta})p(\boldsymbol{\theta}|S)d\theta, \tag{4}$$

To find the likelihood in (4), and to obtain the optimal Gaussian network as in (3), the data are pre-assumed either Gaussian or multinomial [32]. Gaussian BN assumes that the nodes are multivariate Gaussian. That is, expression of node i can be described with mean μ_i and covariance matrix Σ_i of size N × N. The joint probability of the network can be the product of a set of conditional probability distributions given by:

$$p(\mathbf{x}_i|\boldsymbol{a}_i) = \theta_{i,\boldsymbol{a}_i} \sim \mathcal{N}\left(\mu_i + \sum_{j\in\boldsymbol{a}_i}(\mathbf{x}_j - \mu_j)\beta,\ \Sigma_i'\right), \tag{5}$$

where $\Sigma_i' = \Sigma_i - \Sigma_{i,\boldsymbol{a}_i}\Sigma_{\boldsymbol{a}_i}^{-1}\Sigma_{i,\boldsymbol{a}_i}^T$ and β denotes the regression coefficient matrix, Σ_i' is the conditional variance of $\mathbf{x}_i$ given its parent set $\boldsymbol{a}_i$, $\Sigma_{i,\boldsymbol{a}_i}$ is the covariance between observations of $\mathbf{x}_i$ and the variables in $\boldsymbol{a}_i$, and $\Sigma_{\boldsymbol{a}_i}$ is the covariance matrix of $\boldsymbol{a}_i$.

The acyclic condition of BN does not allow feedback among nodes, and feedback is an essential characteristics of real-world GN [33]. Therefore, dynamic Bayesian networks have recently become popular in building GN with time delays mainly due to their ability to model causal interactions as well as feedback regulations [34]. A first-order dynamic BN is defined by a transition network of interactions between a pair of Gaussian networks connecting nodes at time instants τ and $\tau+1$. In time instant $\tau+1$, the parents of nodes are those specified in the time instant τ. Similarly, the Gaussian network of an R-order dynamic system is represented by a Gaussian network comprising $(\mathrm{R}+1)$ consecutive time points and N nodes, or a graph of $(\mathrm{R}+1)\times\mathrm{N}$ nodes. To compute the integral in (4) and to determine the optimal Gaussian network, stochastic search can be used. For instance, variable-order Markov chain Monte Carlo (VMCMC) method is a type of variable-order stochastic search where a Markov chain of Gaussian networks is sampled, which converges to the target distribution. Each new Gaussian network is formed by increasing or decreasing the order of a single edge in the previous Gaussian network in the chain. The acceptance of the new GN S^{new} is given by Metropolis–Hastings ratio:

$$\min\left\{1, \frac{p(S^{new})}{p(S)}.\frac{p(S^{\mathrm{new}}|X)}{p(S|X)}\right\}, \tag{6}$$

The first term is the prior ratio and the second term is the ratio of likelihoods, which can be computed using (5). Detailed derivation of the sampling algorithm is provided in [23]. Finally, to compute $\beta^\tau = \{\beta^\tau_{i,j}\}_{\mathrm{N}\times\mathrm{N}}$, the matrix of regression coefficients corresponding to a delay of τ time points, we take the average weight of each time-delayed edge in different structures sampled during VMCMC after convergence.

Learning of Gaussian networks with time-delays is a challenging task due to the exponential, number of parameters that require large amounts of training data. In this paper, we propose to use of deep belief networks to learn Gaussian networks with time-delays

such that the task of learning each additional time-delay is assigned to a new layer. In the next section, we detail the learning of weights in deep belief networks.

3.3. Deep belief network

A deep belief network is a type of deep neural network that can be viewed as a composite of simple, unsupervised models such as Restricted Boltzmann machines (RBMs) where each RBMs hidden layer serves as the visible layer for the next RBM. RBM is a bipartite graph comprising two layers of neurons: a visible and a hidden layer; it is restricted such that the connections among neurons in the same layer are not allowed.

To compute the weights W of an RBM, we assume that the probability distribution over the input vector $\mathbf{x}$ is given as :

$$p(\mathbf{x}|W) = \frac{1}{Z(W)}\exp^{-\mathrm{E}(\mathbf{x};W)} \tag{7}$$

where $Z(W) = \sum_{\mathbf{x}}\exp^{-\mathrm{E}}(\mathbf{x};W)$ is a normalization constant. Computing the maximum likelihood is difficult as it involves solving the normalization constant, which is a sum of an exponential number of terms. The standard approach is to approximate the average over the distribution with an average over a sample from $p(\mathbf{x}|W)$, obtained by Markov chain Monte Carlo until convergence. This approach is typically slow, has a large variance in gradient estimate, and often is stuck in local minima.

To learn these weights of each RBM layer and maximize the global energy function efficiently, the approximate maximum likelihood Contrastive Divergence (CD)[16] approach can be used which minimizes the difference of two Kullback–Leibler (KL) divergences given by:

$$\mathrm{CD}_k = \mathrm{KL}(p_0\|p_\infty) - \mathrm{KL}(p_k\|p_\infty)$$

$$\text{where}\quad \mathrm{KL}(p_0\|p_\infty) = \sum\nolimits_{\mathbf{x}} p_0(\mathbf{x})\log\frac{p_0(\mathbf{x})}{p(\mathbf{x};W)} \tag{8}$$

where k is the number of sampling steps, p_0 is probability of input sample in step 0. In practice, this method employs each training sample as the starting state vector for the visible layer of an RBM. Here, we use Gibbs sampling over each input sample for a small number of k steps. Hence, in step zero, the states of hidden neurons are sampled from their posterior distribution over the input feature vector at the visible layer of the RBM and the layer weights. For subsequent steps, a reconstruction of the visible layer is produced using the layer weights and the state of hidden neurons. Now the hidden layer is updated using this reconstructed visible layer. If this form of Gibbs sampling is repeated, after a sufficient number of times, it is possible to converge to the equilibrium.

The posterior distribution over the visible nodes may be logistic or normal, corresponding to binary or continuous inputs [35]. The hidden neurons are latent variables that reduce the dimensionality of the problem in most cases. Hence, hidden neurons are always binary, corresponding to presence or absence of a particular feature. To model the binary neurons, all higher layers employ logistic regression. As an example, here we consider the logistic regression model in the visible layer of the RBM. The continuous state $\hat{h}_j$ of the hidden neuron j, with bias b_j, is a weighted sum over all binary visible nodes $\boldsymbol{v}$ and is given by:

$$\hat{h}_j = b_j + \sum_i v_i w_{ij},\ \ h_j = \frac{1}{1+e^{-\hat{h}_j}}. \tag{9}$$

where w_{ij} is the connection weight to hidden neuron j from visible node v_i. The binary state h_j of the hidden neuron is chosen to be zero or one with a probability determined using a sigmoid activation function.

Similarly, in the next iteration, the binary state of each visible neuron are reconstructed and labeled as $\boldsymbol{v}_{recon}$. As described earlier, subsequent updates to hidden neurons are sampled from the posterior distribution over the reconstructed visible layer in the previous step. After a sufficient number of steps, the weights are updated as the difference between the original training sample and the reconstructed visible layer in the final step using:

$$\triangle w_{ij} = \alpha(\langle v_i h_j\rangle_{data} - \langle v_i h_j\rangle_{recon}), \tag{10}$$

where α is the learning rate and $\langle v_i h_j\rangle$ is the expected frequency with which visible unit i and hidden unit j are active together when the visible vectors are sampled from the training set and the hidden neurons are determined by (9). In practice, CD is a very efficient method, with only a single iteration needed to learn the weights.

The learning of the weights in a deep neural network begins with the first layer. Next, in order to learn the weights of the second layer, the states of the hidden neurons in the first layer are determined from each training sample and (9), however the weights are not updated in this step. The features learned in the first RBM layer becomes the input data to the second RBM layer, where learning of weights progresses using CD. In this way, weights of the first RBM layer are used to sample training samples for the second RBM layer and so on. Finally, energy of the DBN can be determined in the final layer as $\mathrm{E} = -\sum_{i,j} v_i h_j w_{ij}$.

DBN are static models and will not be able to capture variable-order time-delays. Hence, in this paper we propose a dynamic form of a DBN where each layer of hidden neurons is capable of learning variable-order delays in the data.

4. Variable-order belief network

Delays in a realistic dynamic system constantly change following a Markov chain. In this section, we introduce a novel variable-order RBM that is capable of learning high-dimensional Gaussian networks with time delays. We achieve this by introducing delays or perturbations to the training samples via a deep stochastic search and a cascade design to learn them.

The perturbations mimic real disturbances or delays in the system. We begin by describing the learning of weights in a VBM. With the first layer of a DBN making up of the proposed VBM, the resulting model is referred to here as a VBN.

The standard RNN output, $\mathbf{x}_l(\tau)$, at time step τ for each layer l is calculated using the following equations :

$$\mathbf{x}_l(\tau) = f(W_{\mathrm{R}}.\mathbf{x}_l(\tau-1) + W_l.\mathbf{x}_{l-1}(\tau)) \tag{11}$$

where W_{R} is the interconnection matrix among hidden neurons and W_l is the weight matrix of connections between hidden neurons and the input nodes, $\mathbf{x}_{l-1}(\tau-1)$ is the input vector at time step τ from layer $l-1$, vectors $\mathbf{x}_l(\tau)$ and $\mathbf{x}_l(\tau-1)$ represent hidden neuron activations at time steps τ and $\tau-1$ and f is the nonlinear activation function. To learn the weights of the RNN, back propagation through time is used where the hidden layer is unfolded in time using duplicate hidden neurons. Each new layer of hidden neurons models a single time delay and is trained using time shifted input data. This approach is inefficient as it gets stuck in local minima easily.

The VBN differs from RNN, in that to train each hidden layer of neurons we extract the corresponding time-delayed features from the original time series. Further, to learn very long delays we progressively increase the order of time-delayed edges in each hidden layer, so that higher hidden layers can learn long delays as cascades of short delays in individual features as in the input layer. Lastly, each layer is learned independent of the other layers in an unsupervised manner, so that the dimensionality of the model and hence the log probability keeps improving with each hidden layer. In what follows, the detailed VBN framework for predicting GN with time-delays is described.

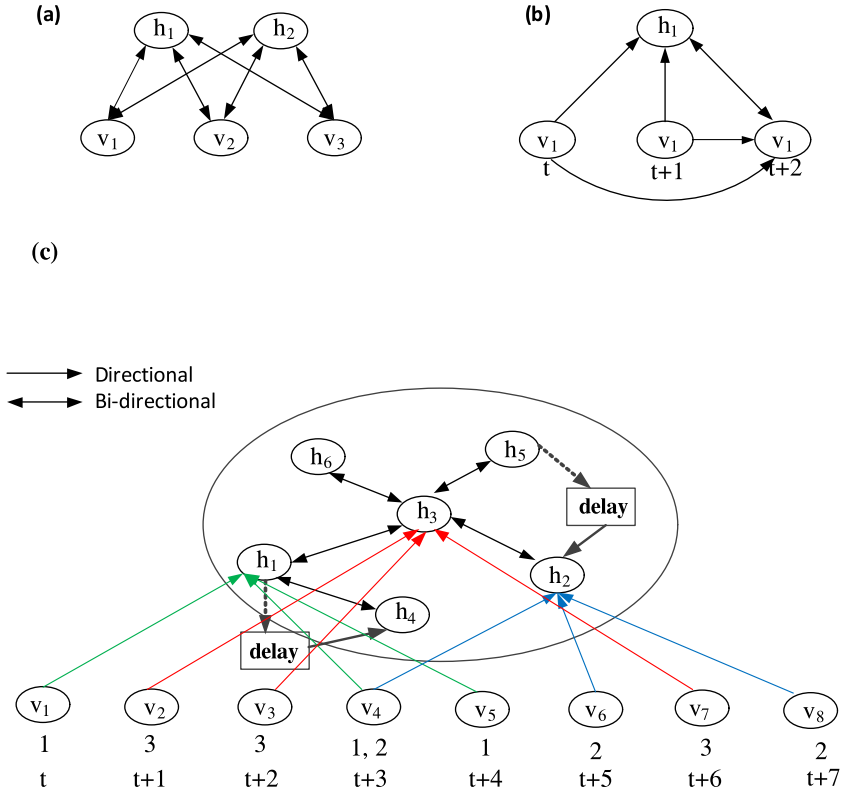

Fig. 2. State-space diagrams for different models. (a) RBM with two hidden neurons and three nodes in the visible layer. (b) Conditional-RBM with single hidden neuron and single visible node [14]. (c) Variable-order BM model, the hidden neurons are inter-connected with delay nodes, each hidden neuron is initialized using a high ML network motif shown as colored edges. For example, hidden neuron 1 is initialized with a network motif made of input nodes 1, 4, and 5. Such a model can account for long-delays among input nodes as cascades of first-order dependencies.

4.1. State-space of a variable-order RBM

The restriction on edges between hidden neurons in an RBM endows it with high learning efficiency on the training data. However, this becomes a drawback when the aim is to model variable-order time delays in the input data. In the proposed VBM, we incorporate edges in the visible layer as an alternative to the classical RBM, to model variable-order dependencies. The new RBM is thus labeled a variable-order RBM. The input features to the VBM extracted using the Gaussian fitness function are binary vectors, where '1' indicates presence of an edge, and '0' indicates absence of an edge in the Gaussian network. Prior to the training of an RBM layer in the DBN, the VMCMC stochastic search is performed on each input feature vector (which serves as a starting state vector) and variable-order delays are introduced as described in [23].

Fig. 2 compares the state-space diagrams of a VBN and a previously proposed conditional-RBM for time-delays [14]. Edges can be directional or bi-directional. All bi-directional edges are learned using Contrastive Divergence given by (10). Fig. 2(a) shows an RBM with two hidden neurons and three input nodes in the visible layer, no edges are available between neurons in the same layer. Fig. 2(b) depicts the conditional-RBM with a single hidden neuron and three visible nodes. The visible node is duplicated over time to model variable-order dependencies, making it computationally very expensive. As explained in [14], the directed edges are learned using the reconstructed state of visible nodes and the past inputs at the node.

Fig. 2(c) shows the variable-order BM model, the hidden neurons are inter-connected with delay nodes, each hidden neuron is initialized using a high ML network motif shown as colored edges. For example, hidden neuron 1 is initialized with a network motif made of input nodes 1, 4, and 5. The time-delayed edges in each network motif are determined using the VMCMC stochastic search. The maximum order of VMCMC is progressively increased during the training process, along with the increasing VBM layers in the VBN. This takes the form of a deep stochastic search and time delays are introduced into the learned features at each hidden neuron in a random manner. Further, such a model can account for long-delays among input nodes as cascades of first-order dependencies.

4.2. Learning of weights in VBN

To train a RNN, we must compute the gradient of the total energy function E over the neurons in the output layer with respect to weights in all the layers below. The output $\mathbf{x}_l = f(\mathbf{x}_{l-1}, W_l)$ of each layer l in an RNN is a function of the inputs $\mathbf{x}_{l-1}$ from previous layer as well as the layer weight W_l. To measure the change in the total energy function when a small change is made to W_l, we can use the chain rule:

$$\frac{\partial \mathrm{E}}{\partial W_l} = \frac{\partial \mathrm{E}}{\partial \mathbf{x}_l} \cdot \frac{\partial f_l(\mathbf{x}_{l-1}, W_l)}{\partial W_l}, \tag{12}$$

where $\partial f_l(\mathbf{x}_{l-1}, W_l)/\partial W_l$ is the Jacobian matrix of f_l with respect to W_l. Each element k, l of this matrix indicates how much the kth output changes when we make a small change to the lth weight. Using the same method, we can compute the gradient of the global energy function E with respect to the previous input layer denoted by $\partial \mathrm{E}/\partial \mathbf{x}_{l-1}$.

To compute the gradient in a recurrent neural networks with input data X and known class labels or outputs y, we can apply the chain rule to compute $\partial \mathrm{E}/\partial W_l$ and $\partial \mathrm{E}/\partial \mathbf{x}_l$. The equations have recurrence, hence to compute all the derivatives, we use a backward sweep called the back-propagation algorithm starting with $\partial \mathrm{E}/\partial \mathbf{x}_{\mathrm{L}} = \partial \mathrm{C}(\mathbf{x}_{\mathrm{L}}, y)/\partial \mathbf{x}_{\mathrm{L}}$, where C is the Euclidean distance

between true labels y and predicted output vector $\mathbf{x}_{\mathrm{L}}$. Further, we can assume that weights at each time point are independent and compute the partial gradient with respect to these weights. The total gradient is equal to the sum of these partial gradients. It can be seen that to model each additional time delay, we need to duplicate the neurons in each hidden layer and is not practical for modeling long delays.

The proposed VBN, which consists of multiple layers of latent or hidden neurons, such that each layer takes the form of an RBM, we directly obtain the initial weights for discrete and distributed time-delays matrices using dynamic Gaussian Bayesian networks. The co-efficient $\beta^{\tau}_{\mathrm{N}\times\mathrm{N}}$ for time delay τ is given by (1) and is determined stochastically as a weighted average over VMCMC samples. Since we learn long delays hierarchically from short delays, the modified state space equation for an R-order interaction in terms of first-order cascades is now given by:

$$x_i(\tau+1) = \begin{matrix} g(x(\tau)) & i=1 \\ x_{i-1}(\tau) & i=2,3,\ldots,\mathrm{R} \end{matrix} \tag{13}$$

where x_i denotes the state of the ith node in the GN at time instant τ and $g(x(\tau))$ is a function over all R nodes at time instant τ. Hence, the state of variable x_i at time instance $\tau+1$ is dependent on the state of variable x_{i-1} at time instant τ. Similarly, variable x_{i-1} is dependent on the state of variable x_{i-2} and so on, resulting in a cascade of nodes and time-delays.

When modeling a change of variables in multiple integrals, the Jacobian is used. In our case, this matrix measures how much a change in the input state of one node can result in changes in the input state of another node in the same layer. The Jacobian of the transformation to state space of (13) from (1) is given by:

$$J_x(\tau+1,1) = \begin{bmatrix} \frac{\partial g(x(\tau))}{\partial x_1(\tau)} & \cdots & \frac{\partial g(x(\tau))}{\partial x_r(\tau)} & \cdots & \frac{\partial g(x(\tau))}{\partial x_{\mathrm{R}}(\tau)} \\ 1 & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \tag{14}$$

where the first row are the partial weights, corresponding to $i=1$ in (13) ; the remaining $\mathrm{R}-1$ rows are for $i=2,3,\ldots,\mathrm{R}$ in (13). These will be one for the previous node and zero otherwise, since it is independent of all other delay nodes. In essence, we are doing a dimensionality reduction from variable delays to first-order [36].

Similar to the RNN defined in (11) the output $\mathbf{x}_l = f(\mathbf{x}_{l-1}, W_l, W_{\mathrm{R}})$ of each layer in a VBN, is a function of additional weights W_{R}, where R is the maximum order of time-delays, for edges between the visible nodes when learning layer l. Given that $J(t+\tau,t) = \partial x_{l-1}(t)/\partial x_{l-1}(t-\tau)$ is the Jacobian matrix expanded over τ time points, we use chain rule to determine the gradient descent equations for each of the three parameters in the activation function $f(\mathbf{x}_{l-1}, W_l, W_{\mathrm{R}})$ as follows :

$$\begin{aligned} \mathrm{E}(\mathbf{x}, W, y) &= \mathrm{C}(\mathbf{x}_{\mathrm{L}}, y) \\ \frac{\partial \mathrm{E}}{\partial \mathbf{x}_{\mathrm{L}}} &= \frac{\partial \mathrm{C}(\mathbf{x}_{\mathrm{L}}, y)}{\partial \mathbf{x}_{\mathrm{L}}} \\ \frac{\partial \mathrm{E}}{\partial W_l} &= \frac{\partial \mathrm{E}}{\partial \mathbf{x}_l} \cdot \frac{\partial f_l(\mathbf{x}_{l-1}, W_l, W_{\mathrm{R}})}{\partial W_l} \\ \frac{\partial \mathrm{E}}{\partial \mathbf{x}_{l-1}} &= \frac{\partial \mathrm{E}}{\partial \mathbf{x}_l} \cdot \frac{\partial f_l(\mathbf{x}_{l-1}, W_l, W_{\mathrm{R}})}{\mathbf{x}_{l-1}} \\ \frac{\partial \mathrm{E}}{\partial W_{\mathrm{R}}} &= \frac{\partial \mathrm{E}}{\partial \mathbf{x}_l} \cdot \sum_{\tau=0}^{\mathrm{R}} \frac{\partial x_{l-1}(t)}{\partial x_{l-1}(t-\tau)} \cdot \frac{\partial f_l(x_{l-1}(\tau), W_{\mathrm{R}})}{\partial W_{\mathrm{R}}} \\ &= \frac{\partial \mathrm{E}}{\partial \mathbf{x}_l} \cdot \sum_{\tau=0}^{\mathrm{R}} \beta^{\tau}_{\mathrm{N}\times\mathrm{N}} \end{aligned} \tag{15}$$

Similar to RNN, we can again apply the chain rule to compute $\partial \mathrm{E}/\partial W_l$ and $\partial \mathrm{E}/\partial \mathbf{x}_l$. Next, to compute the Jacobian matrices of time delay $\partial \mathrm{E}/\partial W_{\mathrm{R}}$ in (15), we can simply use the pre-determined co-efficient using dynamic Gaussian Bayesian networks that provide optimal initial weights to train each layer of neurons.

Since, we slowly increase the number of time delays in each layer different types of time delayed features are learned in each layer. Hence, we do not need additional hidden neurons for time delays and can provide suitable initial weights for the neurons. In addition, due to the cascading of time-delays in the visible layer it can be seen that the very long delays can be modeled as skip-edges where the Jacobian for very long delays is given by (13). The co-efficient matrices for each delay (shaded boxes) also need to be computed only once.

4.3. VBN framework

In this section, we describe the VBN framework. Instead of an RBM, each layer is a variable-order restricted Boltzmann machine (VBM) with a dynamic Gaussian Bayesian network model. On very large networks, the computational bottleneck often lies in the estimation of the likelihood of nodes given their parents, since the number of possible combinations is exponential to the node size of a Gaussian network. Hence, we use the training data to only extract low-order network motifs based on conditional probabilities for nodes given their possible parent set and the number of parents is limited to two or three.

The algorithm can be summarized as two stages: (a) First, multivariate data is used to extract the likelihoods of motifs as conditional probability of nodes given a possible parent set using (5). Edges in motifs with likelihood above threshold γ are used to create the training networks. (b) In the second stage, we use VMCMC algorithms to introduce variable-delays into these training networks and subsequently learn the weights of a DBN. A DBN is capable of learning from bases; hence, we can compute ML parameters of as many motifs as the computational time permits. While, it is desirable to compute all possible ML parameters, a DBN can learn well even from a small subset of ML parameters.

We divide the task of learning each additional time-delay to a new hidden layer. Hence, the l^{th} hidden layer is learned using l-order training motifs. This is achieved by evolving each first-order training sample motif using an l-order VMCMC, the average over sampled structures determines the new edge weights W_r for the corresponding l-order training motif. This is followed by Contrastive Divergence to update the weights W_l of the DBN using the new l-order training motif. Once we have finished learning the l^{th} layer, we increase the order of the VMCMC in the VBM to $l+1$ and the ML parameters of order $r=l+1$ required during stochastic search is precomputed using Algorithm 1 (Algorithm 2, line: 17). For example, after learning the first layer, input motifs are now evolved in a second-order Gaussian network. Now, the second layer of the DBN is learned using Gaussian networks having up-to second order time delays. In this way, a deep stochastic search is achieved.

Algorithm 1 Extracting motifs of order r using Gaussian fitness function

1: Input 1 : Training and testing time series $\boldsymbol{X} = \{X^1, X^2, \ldots, X^{n_d}\}$ for all n_d networks
2: Input 2: Order of time delay r
3: Output : ML parameters of motifs $\boldsymbol{\Theta} = \{\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^{n_d}\}$
4: **for** $s=1$ to n_d **do**
5: $\quad \boldsymbol{\theta}^s = \{\theta_{i,\boldsymbol{a}_i} = p(\mathbf{x}_i(t)|\boldsymbol{a}_i(t-r)), |\boldsymbol{a}_i| \leq d\} \forall i, \forall \boldsymbol{a}_i$ using (5) and time series X^s.
6: $\quad \boldsymbol{\theta}^s = \{\theta_{i,\boldsymbol{a}_i} \geq \gamma\}, \forall \theta_{i,\boldsymbol{a}_i} \in \boldsymbol{\theta}^s$
7: **end for**

Algorithm 2 Deep learning of Gaussian networks with time delays

Input : ML parameters of motifs $\boldsymbol{\Theta} = \{\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^{n_d}\}$ learned in **Algorithm 1** with order of time delay r=1.
Outputs: Gaussian network class label for each test sample
Construct the visible layer as a vector of N input features
Compute number of significant principal components n_l in training data $\boldsymbol{X}$
Construct a minimal network with n_l hidden neurons
Initialize weights of the layer using high ML motifs
Construct the output layer with n_d hidden neuron to decode GN of class s
repeat
  for $s = 1$ to n_d **do**
    for $t = 1$ to $|\theta^s|$ **do**
      Initialize the visible layer with t^{th} training sample in θ^s
      Update W_r the new feature vector using VMCMC and (15)
      Update W_l using CD given by (10) $\forall l$
    end for
  end for
  Compute : ML parameters of motifs $\boldsymbol{\Theta} = \{\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^{n_d}\}$ using **Algorithm 1** with $r = r + 1$.
  Compute change in reconstruction error $\triangle\epsilon$ on training data $\boldsymbol{X}$
  Compute number of significant principal components n_l from layer l training data : $\boldsymbol{X}_l$
  Add another hidden layer of n_l neurons
until Adding a layer does not change $\triangle\epsilon$
Compute ML probability of all training networks using test samples
Classify test samples to Gaussian networks with highest ML probability

Algorithm 2 describes the complete framework for predicting time-delayed networks using a VBN. The motifs with high likelihood are used to create a training database of first-order input features using Algorithm 1. Each training sample is a time sample of N nodes.

To determine the number of layers, we compute the change in visible layer reconstruction error $\triangle\epsilon$ of the VBN on the training samples. This is the root means square error between input training sample and reconstructed sample at each visible node. If there is a significant change in the error $\triangle\epsilon$, a new hidden layer is determined using Algorithm 2. The layer weights are then learned and the classification precision error is recomputed. The above progresses iteratively until further addition of hidden layers does not change the classification precision error significantly, and the optimal configuration is achieved. To determine the optimal number of hidden neurons in a single layer, we consider the number of significant principal components in the training data for that layer.

Each hidden neuron in the final output layer will correspond to the complete Gaussian network of a particular class. The Contrastive Divergence approach will sample edges with high frequency into the upper layers, resulting in the formation of subgraphs at hidden neurons in the first layer, bigger graphs at hidden neurons in second hidden layer and so on. We iterate through the algorithm until there is no significant change in the weights at the l^{th} layer. We keep adding additional layers until there is no further improvement in classification accuracy.

4.4. Computational complexity

The computation complexity of a symmetric weight matrix of n neurons is at least $O((n^2)^3)$. This follows from the three steps for computing the forward error, computing the backward error and the update step. RNN have an additional weight matrix for each time-delay. Hence, if R is the maximum delay the computational complexity becomes $O(((n \times R)^2)^3)$. Previously, RNN with over 1000 layers have been used to model long-delays that are computationally very slow [37].

Similarly, the complexity of the dynamic Gaussian Bayesian network is also exponential in the number of time-delays. Hence, if $|a_i|$ is the cardinality of the parent set, we have to compute the likelihood of $O(R^{|a_i|})$ possible network motifs. A stochastic sampling algorithm called variable-order MCMC can rapidly determine the optimal structure of GN as described in [23]. Further, we are inspired by skip-chain models that approximate long-delays as a cascade of short delays with only quadratic complexity in the length of the delay $O(R^2)$ [6].

The skip-chain is achieved by considering a deep framework where a hierarchy of temporal features of increasing time-delays are learned. The complexity of each layer is only $O((n^2)^3)$. Instead, we assign the task of computing time-delayed parameters to a dynamic Gaussian Bayesian network of a fixed order k. Hence, for each additional layer we only compute the new higher-order network motifs. The computational complexity of a single delay dynamic Bayesian network is always $O(2^{|a_i|})$. The first-order hidden neurons are initialized with only first-order network motifs. The second-order hidden neurons are only initialized with second-order network motifs and so on. Due to the hierarchical structure of deep belief networks. The first-order temporal features may cascade to form second-order temporal features in the second layer and so on when trained. In this way, we divide the task of learning each time-delay to a new layer without any increase in computational cost.

5. Experiments

We applied our proposed algorithm to two real-world problems. The two datasets were real-world data collected from basketball games and the document classification task of "20 Newsgroups" dataset. The first was a small Gaussian network to classify a basketball game as a beginner game or a skilled game using player trajectories captured through a camera [38]. The second was a very high dimensional document dataset. The vocabulary has over 65,000 words and each document is made by words occurring in a sequence. Our method exhibited notable results on both problems. Performance measures such as F-measure[1] and mean square classification error were evaluated using true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN), respectively in the network reconstructed at the visible layer.

5.1. Basketball game

In basketball, it is possible to determine the offensive and defensive teams using player-tracking [39]. Analysis of formations in beginner and skilled games can help in expertise development in players. Previously, deep transfer neural networks were used to classify formations as beginner or skilled [40]. Here, we show that a VBN is able to determine the causal behavior of teams as well as classify formations as beginners and skilled and also as winner or loser.

The trajectories of five players during a basketball game can be represented as a GN network with five nodes. The time-delayed regulations between the players, such as passing of the ball, can be modeled as variable-order edges in the network. The data for the five players from 15 beginner and 15 skilled games is used to compute the likelihood of motifs using (5). The top 20% of motifs sorted by decreasing likelihood are subsequently used to construct the training samples for the VBN as described in Section 4.3. Motifs with high ML in each of the other classes were randomly selected to construct an equal number of samples for the other class.

The optimal configuration determined by our algorithm was a two layer VBN with hidden layers given by [5, 10, 10, 4]. There are 10 recurrent neurons in the first hidden layer, 10 recurrent neurons in the second hidden layer and only four output nodes corresponding to four possible class labels. No further improvement can be observed on adding additional layers.

[1] F-measure $=2\ \frac{Precision \times Recall}{Precision + Recall}$

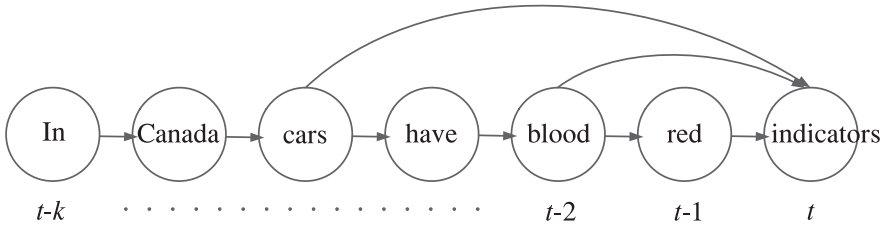

Fig. 3. Sequence model for a sentence that satisfy Markovian condition that state of a word at time t depends on the words in the immediate past $t-1:t-k$ [6].

Table 1
F-measure of correctly classifying a basketball formation by BN, SLFN, RNN and VBN.

Type	BN	SLFN	RNN	**VBN**
Beginner winner	0.65	0.097	0.68	0.82
Beginner loser	0.68	0.465	0.64	0.78
Skilled winner	0.63	0.365	0.57	0.80
Skilled loser	0.71	0.326	0.67	0.81
Total	0.67	0.314	0.64	0.81

Table 1 tabulates the results including TP, FP, precision, recall, and F-measure for classifying the 40 test formations as Beginner or Skilled games as well as Winner or Loser. We assess the results of the trained VBN against the baseline RNN, a single layer feed-forward neural network (SLFN) and Bayesian network classifier. It can be seen in Table 1 that in terms of prediction accuracy, VBN outperformed the accuracy of the baseline algorithms by more than 20% in the skilled games and had above average performance in beginner games. The baselines have high accuracy on beginner games due to presence of few offensive formations, however they perform poorly on skilled games were a diverse sequence of offensive formations are observed. The recurrent neural network (RNN) showed lower F-measure for identifying skilled loser team formations than VBN. The Bayesian network could classify beginner game formations easily however they showed poor performance on skilled games. The SLFN performed the worst as it is unable to capture the temporal changes in player trajectories. The total F-measure of VBN is significantly higher than the baselines. Variance was not reported since it was very small ($< 0.1\%$) even for all the algorithms.

5.2. Classification of documents

Further, the method can also be used on sequence datasets such as words in a sentence. Classification of documents is traditionally done using a pre-defined set of motifs, where the words are the nodes. Weights or ML probability of each motif given its class can be computed from the training data. Documents can then be classified to a group where the sum of weights or ML probabilities over all motifs given their class labels is highest. Here, we use a VBN to determine the optimal motifs and to construct probability distributions over classes. Fig. 3 illustrates the sequence model for a sentence that satisfy Markovian condition that state of a word at time t depends on the words in the immediate past $t-1:t-k$ [6]. The edge 'red → indicators' is a first-order dependency, while the edge 'blood → indicators' is a second-order dependency. To model a long delay dependency such as 'cars → indicators' we can use a skip-edge that uses a cascade of first-order edges to compute the probability of the dependency. Further, when training data is not sufficient an additional level of hidden neurons can use contextual information across the entire network of words resulting in better classification.

To validate the effectiveness of VBN in text clustering, we used the processed version of the "20 Newsgroups" dataset. This benchmark contains 20 groups with over 12,000 documents. As considered in the literature, the data is divided into training (9073 documents) and testing (2241 documents) sets [25]. Each document is made of sentences with maximum length 93 words and variable-order dependencies between the words. The data pre-processing included removing stop words and punctuation marks from the sentences. A convolution neural network with two convolution layers, kernels of width 10, a logistic layer of 300 neurons and 20 output neurons was used to extract features from the sentences. We used the 300 features in the penultimate layer as input to the VBN.

Table 2
F-measure of correctly classifying documents by BN, SLFN, RNN, VBN and CRF (Stanford classifier).

Type	CRF [25]	BN	SLFN	RNN	**VBN**
alt.atheism	0.76	0.61	0.68	0.88	0.95
comp.graphics	0.7	0.45	0.48	0.86	0.88
comp.os.ms-windows.misc	0.75	0.45	0.47	0.89	0.92
comp.sys.ibm.pc.hardware	0.69	0.65	0.67	0.90	0.93
comp.sys.mac.hardware	0.78	0.70	0.75	0.96	0.97
comp.windows.x	0.84	0.73	0.64	0.71	0.95
misc.forsale	0.85	0.38	0.37	0.88	0.95
rec.autos	0.85	0.80	0.91	0.89	0.95
rec.motorcycles	0.92	0.31	0.5	0.83	0.95
rec.sport.baseball	0.85	0.39	0.48	0.87	0.93
rec.sport.hockey	0.91	0.78	0.83	0.94	0.95
sci.crypt	0.92	0.72	0.73	0.88	0.93
sci.electronics	0.71	0.56	0.58	0.78	0.90
sci.med	0.82	0.45	0.58	0.83	0.93
sci.space	0.89	0.53	0.61	0.91	0.92
soc.religion.christian	0.89	0.69	0.67	0.91	0.95
talk.politics.guns	0.78	0.60	0.55	0.91	0.96
talk.politics.mideas	0.9	0.85	0.82	0.94	0.94
talk.politics.misc	0.68	0.38	0.30	0.83	0.92
talk.religion.misc	0.66	0.39	0.56	0.96	1
Total	0.81	0.57	0.62	0.88	0.94

The 300 feature vectors were used to infer likelihood of motifs using (5). The top 20% of motifs sorted by decreasing likelihood are used to initialize the weights of the input layer in the VBN. Motifs of top words in the first class with high ML in the other groups were randomly selected to construct an equal number of samples for the other class as described in Section 4.3 . Next, the VBN algorithm was used to create the complete GN representing the state-space of each group. The optimal configuration of VBN determined by our algorithm was a three layer VBN with hidden layers given by [300, 10, 10, 10, 20] nodes. A third layer of 10 neurons only showed slight improvement on this dataset. The third layer was 20 output nodes for the predicted class network and the network seen in documents that are not from this class, since insignificant improvement can be observed via adding extra layers.

In order to assess the accuracy of the trained VBN, we classified the testing set of documents. Table 2 summarizes the F-measure for classifying each document correctly in a given group. We assess the results of the trained VBN against the baseline Stanford classifier that uses conditional random fields (CRF), a single layer feed-forward neural network (SLFN) and the Bayesian network classifier.

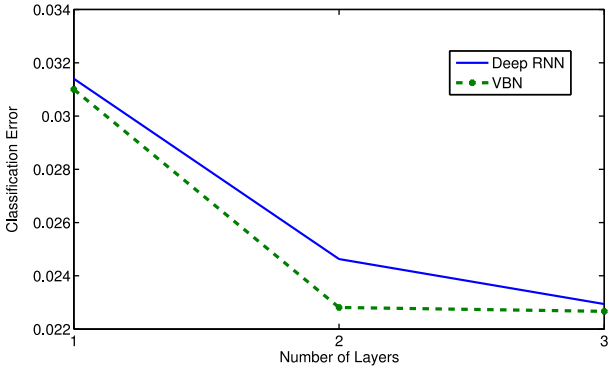

Fig. 4. Change in mean square classification error (MSE) with increasing number of hidden layers.

Variance was not reported since it was very small (<0.1%) even for the baseline algorithm [25]. It can be seen in Table 2 that in terms of prediction accuracy, VBN outperformed the accuracy of the baseline CRF algorithm by more than 20% in the majority of the groups. The recurrent neural network (RNN) performed well on some groups. The Bayesian network showed almost half the accuracy on most groups. The SLFN showed slightly higher accuracy than BN, however it was computationally very slow.

5.3. Effect of number of hidden layers

To determine the number of hidden layers of recurrent neurons we consider the mean square classification error (MSE) on training data. Fig. 4 reports the decrease in MSE with increasing number of hidden layers for the "20 Newsgroups" dataset. It can be seen that MSE decreases significantly on addition of a second hidden layer. Only a slight increase is observed on adding a third hidden layer. Fig. 4 shows the change in MSE for VBN where the weights of hidden neurons are initialized using high ML network motifs with time-delayed edges determined using VMCMC. In the baseline, case without using GBN the MSE is much higher and a significant decrease is only observed in the third hidden layer. Hence, we can conclude that GBN prior is suitable for variable-order belief networks. The classification error will keep improving with more hidden layers, however for this dataset the decrease in error is not significant after three layers.

6. Conclusion

In this paper, we have proposed a variable-order belief network to predict and classify dynamic multivariate Gaussians. Our simulation and experimental study show that VBN outperforms several baseline approaches in terms of prediction accuracy. On the synthetic benchmark network of eight nodes and up to five time point delay, it could achieve almost 30% improvement in prediction accuracy to previous approaches.

RNN for temporal processes require the use of additional hidden states for time-delays, making them inefficient. Because of this, in this paper, VBN uses a cascade of first-order delays to model a long delay and the optimal delays are introduced using a deep stochastic search called VMCMC. By using fewer neurons, we were able to mitigate the problem of over-fitting to training data. VBN intuitively begins with the input network motifs at the first layer, which can be easily derived from the data. These are merged using hidden neurons in the upper layers to learn the complete Gaussian network. In this manner, network motifs are shared across the entire network, and many fewer time samples are hence needed for prediction.

To model time-delays the input layer is initialized using first-order network motifs. The second hidden layer combines first-order network motifs to form second-order network motifs. Similarly, the third hidden layer can be trained using second-order input network motifs and so on.

Acknowledgment

This work is partially supported by the ASTAR Thematic Strategic Research Programme (TSRP) Grant No. 1121720013 and the Center for Computational Intelligence (C2I) at NTU. This work was also supported by the Singapore-MIT Alliance in Computation and Systems Biology and by the MIT Center for Computational Research in Economics and Management Science.

References

[1] N. Friedman, I. Nachman, Gaussian process networks, in: Proceedings of the Sixteenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-00), 2000, pp. 211–219.
[2] D. Heckerman, D. Geiger, D.M. Chickering, Learning bayesian networks: the combination of knowledge and statistical data, Mach. Learn. 20 (3) (1995) 197–243.
[3] Z. Wang, Y. Liu, X. Liu, State estimation for jumping recurrent neural networks with discrete and distributed delays, Neural Networks 22 (1) (2009) 41–48.
[4] E. Cambria, G.-B. Huang, et al., Extreme learning machines, IEEE Intell. Syst. 28 (6) (2013) 30–59.
[5] G.-B. Huang, E. Cambria, K.-A. Toh, B. Widrow, Z. Xu, New trends of learning in computational intelligence, IEEE Comput. Intell. Mag. 10 (2) (2015) 16–17.
[6] M. Galley, A skip-chain conditional random field for ranking meeting utterances by importance, in: Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP 2006), 2006, pp. 364–372.
[7] S. Bock, M. Schedl, Polyphonic piano note transcription with recurrent neural networks, in: Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on, 2012, pp. 121–124.
[8] R. Brueckner, B. Schulter, Social signal classification using deep blstm recurrent neural networks, in: Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on, 2014, pp. 4823–4827.
[9] I. Sutskever, J. Martens, G. Hinton, Generating text with recurrent neural networks, in: Proceedings of the 28th International Conference on Machine Learning (ICML-11), in: ICML '11, 2011, pp. 1017–1024.
[10] M. Chen, Z. Xu, K. Weinberger, F. Sha, Marginalized denoising autoencoders for domain adaptation, in: Proceedings of the 29th International Conference on Machine Learning (ICML-12), in: ICML '12, 2012, pp. 767–774.
[11] X. Glorot, A. Bordes, Y. Bengio, Domain adaptation for large-scale sentiment classification: a deep learning approach, in: In Proceedings of the Twenty-eight International Conference on Machine Learning, ICML, 2011.
[12] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks 61 (0) (2015) 85–117.
[13] D. Scherer, A. Müller, S. Behnke, Evaluation of pooling operations in convolutional architectures for object recognition, in: Proceedings of the 20th International Conference on Artificial Neural Networks: Part III, in: ICANN'10, 2010, pp. 92–101.
[14] G.W. Taylor, G.E. Hinton, S.T. Roweis, Modeling human motion using binary latent variables, in: B. Schölkopf, J. Platt, T. Hoffman (Eds.), Advances in Neural Information Processing Systems 19, MIT Press, Cambridge, MA, 2007, pp. 1345–1352.
[15] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507, doi:10.1126/science.1127647.
[16] G.E. Hinton, Training products of experts by minimizing contrastive divergence, Neural Comput. 14 (8) (2002) 1771–1800.
[17] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, B. Kingsbury, Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups, Signal Processing Mag. IEEE 29 (6) (2012) 82–97.
[18] A. Mohamed, G. Dahl, G. Hinton, Acoustic modeling using deep belief networks, Audio Speech Lang. Process. IEEE Trans. 20 (1) (2012) 14–22.
[19] H. Lee, R. Grosse, R. Ranganath, A.Y. Ng, Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations, in: Proceedings of the 26th Annual International Conference on Machine Learning, in: ICML '09, 2009, pp. 609–616.
[20] E. Cambria, A. Hussain, C. Havasi, C. Eckl, Sentic computing: exploitation of common sense for the development of emotion-sensitive systems, in: A. Esposito, N. Campbell, C. Vogel, A. Hussain, A. Nijholt (Eds.), Development of Multimodal Interfaces: Active Listening and Synchrony, Lecture Notes in Computer Science, Springer, Berlin, 2010, pp. 148–156.
[21] X. Glorot, A. Bordes, Y. Bengio, Domain adaptation for large-scale sentiment classification: a deep learning approach, in: ICML, vol. 27, 2011, pp. 97–110.

[22] E. Cambria, T. Mazzocco, A. Hussain, C. Eckl, Sentic medoids: organizing affective common sense knowledge in a multi-dimensional vector space, in: D. Liu, H. Zhang, M. Polycarpou, C. Alippi, H. He (Eds.), Advances in Neural Networks, Lecture Notes in Computer Science, vol. 6677, Springer-Verlag, Berlin, 2011, pp. 601–610.
[23] J. Rajapakse, I. Chaturvedi, Gene regulatory networks with variable-order dynamic bayesian networks, in: Neural Networks (IJCNN), The 2010 International Joint Conference on, 2010, pp. 1–5.
[24] M. Bee, Modelling credit default swap spreads by means of normal mixtures and copulas, Appl. Math. Finance 11 (2) (2004) 125–146.
[25] Wikipedia, Stanford classifier, 20 newsgroup dataset @ONLINE, 2012. URL http://nlp.stanford.edu/wiki/Software/Classifier
[26] E. Cambria, Affective computing and sentiment analysis, IEEE Intell. Syst. 31 (2) (2016) 102–107.
[27] S. Poria, A. Gelbukh, E. Cambria, A. Hussain, G.-B. Huang, EmoSenticSpace: a novel framework for affective common-sense reasoning, Knowledge Based Syst. 69 (2014) 108–123.
[28] E. Cambria, A. Hussain, T. Durrani, C. Havasi, C. Eckl, J. Munro, Sentic computing for patient centered application, in: IEEE ICSP, Beijing, 2010, pp. 1279–1282.
[29] E. Cambria, B. Schuller, B. Liu, H. Wang, C. Havasi, Statistical approaches to concept-level sentiment analysis, IEEE Intell. Syst. 28 (3) (2013) 6–9.
[30] A. Prinzie, D. Van den Poel, Dynamic Bayesian Networks for Acquisition Pattern Analysis: A Financial-Services Cross-Sell Application New Frontiers in Applied Data Mining, in: Lecture Notes in Computer Science, vol. 5433, Springer Berlin / Heidelberg, 2009, pp. 123–133.
[31] A.C. Rencher, Methods of Multivariate Analysis, 2nd, Wiley, 2002.
[32] E. Castillo, J.M. Gutierrez, A.S. Hadi, Expert Systems and Probabilistic Network Models, Springer-Verlag New York, Inc., 1996.
[33] J. Wagner, G. Stolovitzky, Stability and time-delay modeling of negative feedback loops, Proc. IEEE 96 (8) (2008) 1398–1410.
[34] N. Friedman, K. Murphy, S. Russell, Learning the structure of dynamic probabilistic networks, in: Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98), 1998. 139–14
[35] G. Dahl, M.A Ranzato, A.R Mohamed, G.E. Hinton, Phone recognition with the mean-covariance restricted boltzmann machine, in: Advances in Neural Information Processing Systems 23, 2010, pp. 469–477.
[36] L. Tsungnan, B.G. Horne, P. Tino, C.L. Giles, Learning long-term dependencies in narx recurrent neural networks, Neural Networks IEEE Trans. 7 (6) (1996) 1329–1338.
[37] F. Gers, J. Schmidhuber, Lstm recurrent networks learn simple context-free and context-sensitive languages, Neural Networks IEEE Trans. 12 (6) (2001) 1333–1340.
[38] K.Y.H. Chow Jia Yi, I. Chaturvedi, An analysis of herding behaviours in basketball as a function of skill level., ECSS, 2013.
[39] H.-T. Chen, C.-L. Chou, T.-S. Fu, S.-Y. Lee, B.-S. P. Lin, Recognizing tactic patterns in broadcast basketball video using player trajectory, J. Visual Commun. Image Represent. 23 (6) (2012) 932–947.
[40] I. Chaturvedi, Y.-S. Ong, R.V. Arumugam, Deep transfer learning for classification of time-delayed gaussian networks, Signal Process. 110 (0) (2015) 250–262.