Nonlinear Relational Markov Networks with an Application to the Game of Go

Tapani Raiko

Neural Networks Research Centre, Helsinki University of Technology,
P.O.Box 5400, FI-02015 HUT, Espoo, FINLAND
Tapani.Raiko@hut.fi

Abstract. It would be useful to have a joint probabilistic model for a general relational database. Objects in a database can be related to each other by indices and they are described by a number of discrete and continuous attributes. Many models have been developed for relational discrete data, and for data with nonlinear dependencies between continuous values. This paper combines two of these methods, relational Markov networks and hierarchical nonlinear factor analysis, resulting in joining nonlinear models in a structure determined by the relations in the data. The experiments on collective regression in the board game go suggest that regression accuracy can be improved by taking into account both relations and nonlinearities.

1 Introduction

Growing amount of data is collected every day in all fields of life. For the purpose of automatic analysis, prediction, denoising, classification etc. of data, a huge number of models have been created. It is natural that a specific model for a specific purpose works often the best, but still, a general method to handle any kind of data would be very useful. For instance, if an artificial brain has a large number of completely separate modules for different tasks, the interaction between the modules becomes difficult. Probabilistic modelling provides a well-grounded framework for data analysis. This paper describes a probabilistic model that can handle data with relations as well as discrete and continuous values with nonlinear dependencies.

Terminology: Using Prolog notation, we write $\mathrm{knows}(\mathrm{alex}, \mathrm{bob})$ for stating a fact that the knows relation holds between the objects alex and bob, that is, Alex knows Bob. The arity of the relation tells how many objects are involved. The knows relation is binary, that is, between two objects, but in general relations can be of any arity. The atom $\mathrm{knows}(\mathrm{alex}, B)$ matches all the instances where the variable B represents an object known by Alex. In this paper, the terms are restricted to constants and variables, that is, compound terms such as $\mathrm{thinks}(A, \mathrm{knows}(B, A))$ are not considered. For every relation that is logically true, there are associated attributes $\mathbf{x}$, say a class label or a vector of real numbers. The attributes $\mathbf{x}(\mathrm{knows}(A, B))$ describe how well A knows B and whether A likes or dislikes B. The attribute vector $\mathbf{x}(\mathrm{con}(A))$ describes what kind of a consumer the person A is. Given a relational database describing relationships between people and their consuming habits, we might study the dependencies that might be found. For instance, some people cloth like their idols, and nonsmokers tend to be

KNOWS

who	whom	how
Alex	Bob	friend
Bob	Carl	neighbour
Carl	Alex	colleague

CONSUMER

who	smoker	...
Alex	no	...
Bob	no	...
Carl	no	...

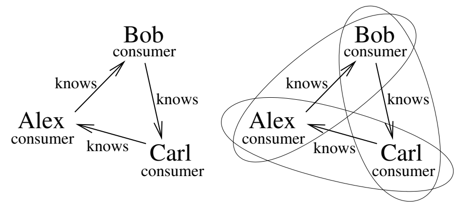

Fig. 1. Consider a relational database describing the relationships and consumer habits of three people. The two tables are shown on the left. On the right, the database is represented graphically, with the occurrences of the template $(\mathrm{con}(A), \mathrm{knows}(A, B), \mathrm{con}(B))$ marked with ovals on the very right.

friends with nonsmokers. The modelling can be done for instance by finding all occurrences of the template $(\mathrm{con}(A), \mathrm{knows}(A, B), \mathrm{con}(B))$ in the data and studying the distribution of the corresponding attributes. The situation is depicted in Figure 1.

Bayesian networks[6] are popular statistical models based on a directed graph. The graph has to be acyclic, which is in line with the idea that the arrows represent causality: an occurrence cannot be its own cause. In relational generalisations of Bayesian networks [7], the graphical structure is determined by the data. Often it can be assumed that the data does not contain cycles, for instance in the case when the direction of the arrows is always from the past to the future. Sometimes the data has cycles, like in Figure 1. Markov networks [6], on the other hand, are based on undirected graphical models. A Markov network does not care whether A caused B or vice versa, it is interested only whether there is a dependency or not.

2 Model Description

This section describes the models that are combined into nonlinear relational Markov networks.

2.1 Hierarchical Nonlinear Factor Analysis (HNFA)

In (linear) factor analysis, continuous valued observation vectors $\mathbf{x}(t)$ are generated from unknown factors (or sources) $\mathbf{s}(t)$, a bias vector $\mathbf{b}$, and noise $\mathbf{n}(t)$ by $\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{b} + \mathbf{n}(t)$. The factors and noise are assumed to be Gaussian and independent. The index t may represent time or the object of the observation. The mapping $\mathbf{A}$, the factors, and parameters such as the noise variances are found using Bayesian learning. Factor analysis is close to principal component analysis (PCA). The unknown factors may represent some real phenomena, or they may just be auxillary variables for inducing a dependency between the observations.

Hierarchical nonlinear factor analysis (HNFA) [11] generalises factor analysis by adding more layers of factors that form a multi-layer perceptron type of a network. In this paper, there are two layers of factors $\mathbf{h}$ and $\mathbf{s}$, and the mappings are:

$$\mathbf{h}(t) = \mathbf{B}\mathbf{s}(t) + \mathbf{b} + \mathbf{n}_h(t) \tag{1}$$

$$\mathbf{x}(t) = \mathbf{A}\mathbf{f}[\mathbf{h}(t)] + \mathbf{C}\mathbf{s}(t) + \mathbf{a} + \mathbf{n}_x(t)\,, \tag{2}$$

where the nonlinearity $\mathbf{f}(\xi) = \exp(-\xi^2)$ operates on each element separately. HNFA can easily be implemented using the Bayes Blocks software library [10, 12]. The update rules are automatically derived in a manner shortly described below.

The unknown variables $\boldsymbol{\theta}$ (factors, mappings, and the parameters) are learned from data with variational Bayesian learning [4]. A parametric distribution $q(\boldsymbol{\theta})$ over the unknown variables $\boldsymbol{\theta}$ is fitted to the true posterior distribution $p(\boldsymbol{\theta} \mid \boldsymbol{X})$ where the matrix $\boldsymbol{X}$ contains all the observations $\mathbf{x}(t)$. The misfit is measured by Kullback-Leibler divergence $D(\cdot \parallel \cdot)$. An additional term $-\log p(\boldsymbol{X})$ is included to avoid calculation of the model evidence term $p(\boldsymbol{X}) = \int p(\boldsymbol{X}, \boldsymbol{\theta}) d\boldsymbol{\theta}$. The cost function is

$$\mathcal{C} = D(q(\boldsymbol{\theta}) \parallel p(\boldsymbol{\theta}|\boldsymbol{X})) - \log p(\boldsymbol{X}) = \left\langle \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{X}, \boldsymbol{\theta})} \right\rangle , \tag{3}$$

where $\langle \cdot \rangle$ denotes the expectation over distribution $q(\boldsymbol{\theta})$. Note that since $D(q \parallel p) \geq 0$, it follows that the cost function provides a lower bound for the model evidence $p(\boldsymbol{X}) \geq \exp(-\mathcal{C})$. The posterior approximation $q(\boldsymbol{\theta})$ is chosen to be Gaussian with a diagonal covariance matrix.

It is possible, though slightly impractical, to model also discrete values in HNFA by using the discrete variable with a soft-max prior [12]. In the binary case, the ith component of $\mathbf{x}(t)$ is left as a latent auxiliary variable, and an observed binary variable $y(t)$ is conditioned by $p(y(t) = 1 \mid x_i(t)) = \frac{\exp x_i(t)}{1+\exp x_i(t)}$. The general discrete case follows analogously requiring more than one auxiliary component of $\mathbf{x}(t)$. The experiments in Section 3 use a thousand copies of a binary variable having the same conditional probability. They can be united into one variable by multiplying its cost by one thousand. Observing 800 ones and 200 zeros corresponds to fixing the variable to a distribution of 0.8 times one and 0.2 times zero.

2.2 Relational Markov Networks (RMN)

A relational Markov network (RMN) [9] is a model for data with relations and discrete attributes. It is specified by a set of clique templates $\mathbf{C}$ and corresponding potentials $\boldsymbol{\Phi}$. Using the example in the introduction, a model can be formed by defining a single clique template $C = (\mathrm{con}(A), \mathrm{knows}(A, B), \mathrm{con}(B))$ and the corresponding potential ϕ_C over $\mathbf{x}(C)$ which is (a subset of) the concatenation of attribute vectors $\mathbf{x}(\mathrm{con}(A))$, $\mathbf{x}(\mathrm{knows}(A, B))$, and $\mathbf{x}(\mathrm{con}(B))$. Given a relational database, the RMN produces an unrolled Markov network over all the attributes $\boldsymbol{X}$. The cliques $c \in C(\mathcal{I})$ instantiated by a template C share the same clique potential ϕ_C. The combined probabilistic model is $p(\boldsymbol{X}) = \frac{1}{Z} \prod_{C \in \mathbf{C}} \prod_{c \in C(\mathcal{I})} \phi_C(\mathbf{x}(c))$, where Z is a normalisation constant and $C(\mathcal{I})$ contains all the instantiations of the template C. In general, a template can be any boolean formula over the relations.

The general inference task is to compute the posterior distribution over all the variables $\boldsymbol{X}$. The network induced by data can be very large and densely connected, so exact inference is often intractable [9]. The belief propagation (BP) algorithm [6] is guaranteed to converge to the correct marginal probabilities only for singly connected Markov networks, but it is used as a good approximation also in the loopy case. The learning task, or the estimation of the potentials $\boldsymbol{\Phi}$ is done using the maximum a posteriori criterion. It requires an iterative algorithm alternating between updating the parameters of the potentials and running the inference algorithm on the unrolled Markov network.

2.3 Nonlinear Relational Markov Networks (NRMN)

In nonlinear relational Markov networks (NRMN), the clique potentials are replaced by a probability density function for continuous values, in this case HNFA[1] The combination of these two methods is not completely straightforward. For instance, marginalisation required by the BP algorithm is often difficult with nonlinear models. Also, algorithmic complexity needs to be considered, since the model will be quite demanding. One of the key points is to use probability densities p in place of potentials $\boldsymbol{\Phi}$. Then, overlapping templates give multiple probability functions for some variable and they are combined using the product-of-experts combination rule described below.

Combination Rules: One of the non-trivialities in making relational extensions of probabilistic models is the so called combination rule [7]. When the structure of the graphical model is determined by the data, one cannot know in advance how many links there are for each node. One solution is to use combination rules such as the noisy-or. Combination rule transforms a number of probability functions into one. Noisy-or does not generalise well to continuous values, but two alternatives are introduced below.

Using a Markov network and the BP algorithm corresponds to using probability densities as potentials and the *maximum entropy* combination rule. The probability densities $p_C(\mathbf{x}(C))$ are combined to form the joint probability distribution by maximising the entropy of $p(\boldsymbol{X})$ given that all instantiations of $p_C(\mathbf{x}(C))$ coalesce with the corresponding marginals of $p(\boldsymbol{X})$. For singly connected networks, this means that the joint distribution is $p(\boldsymbol{X}) = \prod_c p_c(\mathbf{x}(c)) / \prod_k p_k(\mathbf{x}(k))$, where k runs over pairs of instantiations of templates and $\mathbf{x}(k)$ contains the shared attributes in those pairs. Marginalisation of nonlinear models cannot usually be done exactly and therefore one should be very careful with the denominator. Also, one should take care in handling loops.

In the *product-of-experts* (PoE) combination rule, the logarithm of the probability density of each variable is the average of the logarithms of the probability functions that the variable is included in: $p(x) \propto \sqrt[n]{\prod_{C \in \mathbf{C}} \prod_{c \in C(\mathcal{I})} p_C(x)}$ for all $x \in \boldsymbol{X}$, where only those n instantiated templates c that contain x, are considered. PoE is easy to implement in the variational Bayesian framework because the term in the cost function (3) can be split into familiar looking terms. Consider the combination of two probability functions $p_1(x)$ and $p_2(x)$ (that are assumed to be independent):

$$\left\langle \log \frac{q(x)}{\sqrt{p_1(x)p_2(x)}} \right\rangle = \frac{\left\langle \log \frac{q(x)}{p_1(x)} \right\rangle + \left\langle \log \frac{q(x)}{p_2(x)} \right\rangle}{2}. \tag{4}$$

A characteristic of PoE is that implicit weighting happens in some sense automatically. When one of the experts gives a distribution with a large variance and another one with a small variance, the combination is close to the one with small variance.

Inference in Loopy Networks: Inferring unobserved attributes in a database is in this case an iterative process which should end up in a cohesive whole. Information can traverse through multiple relations. [2] The basic element in the inference algorithm

[1] One could also think in terms of e.g. a mixture model.

[2] In mixture of experts (MoE), it is enough when only one of the experts explains the data even if all the other disagree. The ignored experts will not pass information on. This explains why the author did not consider MoE as a combination rule.

of Bayes Blocks is the update of the posterior approximation $q(\cdot)$ of a single unknown variable, assuming the rest of the distribution fixed. The update is done such that the cost function (3) is minimised. One should note that when the distribution over the Markov blanket of a variable is fixed, the local update rules apply, regardless of any loops in the network. Therefore the use of local update rules is well founded, that is, local inference in a loopy network does not bring any additional heuristicity to the system. Also, since the inference is based on minimising a cost function, the convergence is guaranteed, unlike in the BP algorithm.

Learning: The learning or parameter estimation problem is to find the probability functions associated with the given clique templates $\mathbf{C}$. Now that we use probability functions instead of potentials, it is possible in some cases to separate the learning problem into parts. For each template $C \in \mathbf{C}$, find the appropriate instantiations $c \in C(\mathcal{I})$ and collect the associated attributes $\mathbf{x}(c)$ into a table. Learn a HNFA model for this table ignoring the underlying relations. This divide-and-conquer strategy makes learning comparatively fast, because all the interaction is avoided. There are some cases that forbid this. If the data contains missing values, they need to be inferred using the method in the previous paragraph. Also, it is possible to train experts cooperatively rather than separately [3].

Clique Templates: In data mining, so called frequent sets are often mined from binary data. Frequent sets are groups of binary variables that get the value 1 together often enough to be called frequent. The generalisation of this concept to continuous values could be called the *interesting sets*. An interesting set contains variables that have such strong mutual dependencies that the whole is considered interesting. The methodology of inductive logic programming could be applied to finding interesting clique templates. The definition of a measure for interestingness is left as future work. Note that the divide-and-conquer strategy described in the previous paragraph becomes even more important if one needs to consider different sets of templates. One can either learn a model for each template separately and then try combinations with the learned models, or try a combination of templates and learn cooperatively the models for them. Naturally the number of templates is much smaller than the number of combinations and thus the first option is computationally much cheaper.

So what are meaningful candidates for clique templates? For instance, the template $(\mathrm{con}(A), \mathrm{con}(B))$ does not make much sense. Variables A and B are not related to each other, so when all pairs are considered, $\mathrm{con}(A)$ and $\mathrm{con}(B)$ are always independent and thus uninteresting by definition. In general, a template is uninteresting, if it can be split into two parts that do not share any variables. When considering large templates, the number of involved attributes grows large as well, which makes learning more involved. An interesting possibility is to make a hierarchical model. When a large template contains others as subtemplates, one can use the factors $\mathbf{s}$ in Eq. (1, 2) of the subtemplate as the attributes for the large template. The factors already capture the internal structure of the subtemplates and thus the probabilistic model of the large template needs only to concentrate on the structure between its subtemplates.

3 Experiments with the Game of Go

Game of Go: Go is an ancient oriental board game. Two players, black and white, alternately place stones on the empty points of the board until they both pass. The standard board is 19 by 19 (i.e. the board has 19 lines by 19 lines), but 13-by-13 and 9-by-9 boards can also be used. The game starts with an empty board and ends when it is divided into black and white areas. The one who has the larger area wins. Stones of one colour form a block when they are 4-connected. Empty points that are 4-connected to a block are called its liberties. When a block loses its last liberty, it is removed from the board. After each move, surrounded opponent blocks are removed and only after that, it is checked whether the block of the played move has liberties or not. There are different rulesets that define more carefully what a "larger area" is, whether suicide is legal or not, and how infinite repetitions are forbidden.

Computer Go: Of all games of skill, go is second only to chess in terms of research and programming efforts spent. While go programs have advanced considerably in the last 10-15 years, they can still be beaten easily by human players of moderate skill. [5] One of the reasons behind the difficulty of static board evaluation is the fact that there are stones on board that will eventually be captured, but not in near future. In many cases experienced go players can classify these dead stones with ease, but using a simple look-ahead to determine the status of stones is not always feasible since it might take dozens of moves to actually capture the stones.

Experiment Setting: The goal of the experiments was to learn to determine the status of the stones without any lookahead. An example situation is given in Figure 2. The data was generated using a go-playing program called Go81 [8] set on level 1 and using randomness to have variability. By playing the game from the current position to the end a thousand times, one gets an estimate of who is going to own each point on the board. Information on the board states was saved to a relational database with two tables for learning an NRMN. The $\mathbf{x}(\mathrm{block}(A))$ contains the colour, the number of liberties, the size, distances from the edges, influence features in the spirit of [1], and finally the count of how many times the block survives in the 1000 possible futures. The $\mathrm{ally}(A, B)$ and $\mathrm{enemy}(A, B)$ contain a measure of strength of the connection between the blocks A and B estimated using similar influence features [1]. Only the pairs with a strong enough influence on each other (> 0.02) were included. One thousand 13-by-13 board positions after playing 2 to 60 moves were used for learning.

Two clique templates, $((\mathrm{block}(A), \mathrm{ally}(A, B), \mathrm{block}(B))$ and an analogous one for enemy, were used. HNFA models were taught with 28 attributes of the two blocks and the pair. The dimensionality of the $\mathbf{s}$ layer was 8. The learning algorithm pruned the dimensionality of the $\mathbf{h}$ layer to 41 for allies and to 47 for enemies. The models were learned for 500 sweeps through the data. A linear factor analysis model was learned with the same data for comparison. A separate collection of 81 board positions with 1576 blocks was used for testing. The status of each block was now hidden from the model and only the other attributes were known. With inference in the network, the status were collectively regressed. As a comparison experiment, the inferences were also done separately, and combined only in the end. Inference required from four to thirty iterations to converge. As a postprocessing step, the regressed survival probabilities $\hat{x}$

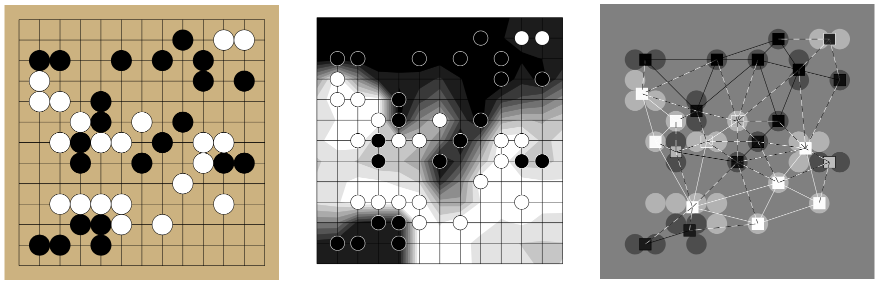

Fig. 2. The leftmost subfigure shows the board of a go game in progress. In the middle, the expected owner of each point is visualised with the shade of grey. For instance, the two white stones in the upper right corner are very likely to be captured. The rightmost subfigure shows the blocks with their expected owner as the colour of the square. Pairs of related blocks are connected with a line which is dashed when the blocks are of opposing colours.

were modified with a simple three-parameter function $\hat{x}_{new} = a\hat{x}^b + c$ and the three parameters that gave the smallest error for each setting, were used.

Results: The table below shows the root mean square (rms) errors for inferring the survival probabilities of the blocks in test cases. They can be compared to the standard deviation 0.2541 of the probabilities.

rms error	Linear	Nonlinear
Separate regression	0.2172	0.2052
Collective regression	0.2171	0.2037

As expected, nonlinear models were better than linear ones and collective regression was better than separate regression.

4 Discussion and Conclusion

A traditional Markov network was applied for statically determining the status of the go board in [2]. Games played by people were used as data. Humans play the game better, but still, this approach has an important downside. The data contains only one possible future for each board position whereas a computer player can produce many possible futures. At the learning stage, all those futures can be used together for the computational price of one. Also, stones that are provably determined to be captured under optimal play (*dead*), might still be useful: By threathening to revive them, the player can gain elsewhere. When data is gathered with unoptimal play, the stones are marked as *not quite dead*, which might be desirable.

NRMN includes a probabilistic model only for the attributes and not for the logical relations. Link uncertainty means that one models the possibility of a certain relation to exist or not. Actually one can model link uncertainty using just the proposed methodology. All the uncertain relations are assumed to be logically true and an additional binary attribute is included to mark whether the link exists or not. One only needs to take into account that when this binary attribute gets the value zero, the dependencies between the other attributes are not modelled. Also, time series data can be represented using relations $\mathrm{obs}(T)$ for the observations at time T and $\mathrm{ensues}(T1, T2)$ to denote that the time indices $T1$ and $T2$ are adjacent. These two examples give light to the generality of the proposed method. In [12], HNFA is augmented with a variance model. Modelling variances would be important also in the NRMN setting, because then each expert would produce an estimate of its accuracy and thus implicitly a weight compared to other experts. In [9], relational Markov networks were constructed to be discriminative so that the model is specialised to classification. The same could be applied here.

Conclusion: A model was proposed for data containing both relations and nonlinear dependencies. The model was built by combining two state-of-the art probabilistic models, hierarchical nonlinear factor analysis and relational Markov networks by using the product-of-experts combination rule. Many simplifying assumptions were made, such as diagonality of the posterior covariance matrix, and separate learning of experts. Also, learning the model structure (the set of clique templates) was left as future work. Experiments with the game of go give promise for the proposed methodology.

Acknowledgements: The author thanks Kristian Kersting, Harri Valpola, Markus Harva, and Alexander Ilin for useful discussions and comments. This research has been funded by the Finnish Centre of Excellence Programme (2000-2005) under the project New Information Processing Principles, and by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the author's views.

References

1. B. Bouzy. Mathematical morphology applied to computer go. *IJPRAI*, 17(2), 2003.
2. T. Graepel D. Stern and D. MacKay. Modelling uncertainty in the game of Go. In *Proc. of the Conference on Neural Information Processing Systems*, Vancouver, December 2004.
3. G.E. Hinton. Modelling high-dimensional data by combining simple experts. In *Proc. AAAI-2000*, Austin, Texas.
4. M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An introduction to variational methods for graphical models. In M. Jordan, editor, *Learning in Graphical Models*, pages 105–161. The MIT Press, Cambridge, MA, USA, 1999.
5. M. Müller. Computer Go. *Special issue on games of Artificial Intelligence Journal*, 2001.
6. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.
7. L. De Raedt and K. Kersting. Probabilistic logic learning. *ACM-SIGKDD Explorations, special issue on Multi-Relational Data Mining*, 5(1):31–48, July 2003.
8. T. Raiko. The go-playing program called Go81. In *Proceedings of the Finnish Artificial Intelligence Conference, STeP 2004*, pages 197–206, Helsinki, Finland, 2004.
9. B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proc. Conference on Uncertainty in Artificial Intelligence (UAI02)*, Edmonton, 2002.
10. H. Valpola, A. Honkela, M. Harva, A. Ilin, T. Raiko, and T. Östman. Bayes blocks software library. `http://www.cis.hut.fi/projects/bayes/software/`, 2003.
11. H. Valpola, T. Östman, and J. Karhunen. Nonlinear independent factor analysis by hierarchical models. In *Proc. ICA2003*, pages 257–262, Nara, Japan, 2003.
12. H. Valpola, T. Raiko, and J. Karhunen. Building blocks for hierarchical latent variable models. In *Proc. ICA2001*, pages 710–715, San Diego, USA, 2001.