LNAI 11726

Gabriele Kern-Isberner
Zoran Ognjanović (Eds.)

Symbolic and Quantitative Approaches to Reasoning with Uncertainty

15th European Conference, ECSQARU 2019
Belgrade, Serbia, September 18–20, 2019
Proceedings

Springer

Lecture Notes in Artificial Intelligence 11726

Subseries of Lecture Notes in Computer Science

More information about this series at http://www.springer.com/series/1244

Gabriele Kern-Isberner · Zoran Ognjanović (Eds.)

Symbolic and Quantitative Approaches to Reasoning with Uncertainty

15th European Conference, ECSQARU 2019
Belgrade, Serbia, September 18–20, 2019
Proceedings

Springer

Editors
Gabriele Kern-Isberner
Technische Universität Dortmund
Dortmund, Germany

Zoran Ognjanović
Mathematical Institute of the Serbian
Academy of Sciences and Arts
Belgrade, Serbia

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Artificial Intelligence
ISBN 978-3-030-29764-0 ISBN 978-3-030-29765-7 (eBook)
https://doi.org/10.1007/978-3-030-29765-7

LNCS Sublibrary: SL7 – Artificial Intelligence

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The biennal ECSQARU conference is a major forum for advances in the theory and practice of reasoning under uncertainty. Contributions are provided by researchers in advancing the state of the art and practitioners using uncertainty techniques in applications. The scope of the ECSQARU conferences encompasses fundamental topics as well as practical issues, related to representation, inference, learning, and decision making both in qualitative and numeric uncertainty paradigms.

Previous ECSQARU events were held in Lugano (2017), Compiegne (2015), Utrecht (2013), Belfast (2011), Verona (2009), Hammamet (2007), Barcelona (2005), Aalborg (2003), Toulouse (2001), London (1999), Bonn (1997), Fribourg (1995), Granada (1993), and Marseille (1991).

The 15th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2019) was held in Belgrade, Serbia, during September 18–20, 2019. The 41 papers in this volume were selected from 62 submissions, after a rigorous peer-review process by the members of the Program Committee and some external reviewers. Each submission was reviewed by at least 2, and on the average 3.1, reviewers. ECSQARU 2019 also included invited talks by outstanding researchers in the field: Fabio Gagliardi Cozman (University of São Paulo), Lluís Godo (Artificial Intelligence Research Institute IIIA, Spanish National Research Council CSIC), and Francesca Toni (Imperial College London).

We would like to thank all those who submitted papers, the members of the Program Committee and the external reviewers for their valuable reviews, and the members of the local Organizing Committee for their contribution to the success of the conference. Financial support from the Ministry of Education, Science and Technological Development of the Republic of Serbia, as well as operational support from the Serbian Academy of Sciences and Arts Council was greatly appreciated. We are also grateful to Springer Nature for granting a Best Paper Award of the conference, and for the smooth collaboration when preparing the proceedings. Moreover, EasyChair proved to be a convenient platform for handling submissions, reviewing, and final papers for the proceedings of ECSQARU 2019, which was greatly appreciated.

July 2019

Gabriele Kern-Isberner
Zoran Ognjanović

Organization

Conference and PC Chairs

Gabriele Kern-Isberner	Technische Universität Dortmund, Germany
Zoran Ognjanović	Mathematical Institute of the Serbian Academy of Sciences and Arts, Serbia

Program Committee

Leila Amgoud	IRIT, CNRS, France
Alessandro Antonucci	IDSIA, Switzerland
Ringo Baumann	Leipzig University, Germany
Christoph Beierle	University of Hagen, Germany
Vaishak Belle	The University of Edinburgh, UK
Concha Bielza	Universidad Politécnica de Madrid, Spain
Isabelle Bloch	LTCI, Télécom Paris, France
Giovanni Casini	University of Luxembourg, Luxembourg
Laurence Cholvy	ONERA-Toulouse, France
Giulianella Coletti	University of Perugia, Italy
Inés Couso	University of Oviedo, Spain
Fabio G. Cozman	University of São Paulo, Brazil
Fabio Cuzzolin	Oxford Brookes University, UK
Luis M. De Campos	University of Granada, Spain
Thierry Denoeux	Université de Technologie de Compiègne, France
Sebastien Destercke	CNRS, UMR Heudiasyc, France
Dragan Doder	IRIT, Université Paul Sabatier, France
Florence Dupin de St-Cyr	IRIT, Université Paul Sabatier, France
Zied Elouedi	Institut Superieur de Gestion de Tunis, Tunisia
Patricia Everaere	CRIStAL - Université Lille, France
Alessandro Facchini	IDSIA, Switzerland
Eduardo Fermé	Universidade da Madeira, Portugal
Tommaso Flaminio	Artificial Intelligence Research Institute, IIIA-CSIC, Spain
Laurent Garcia	LERIA, Université d'Angers, France
Laura Giordano	DISIT, Università del Piemonte Orientale, Italy
Lluís Godo	Artificial Intelligence Research Institute, IIIA-CSIC, Spain
Anthony Hunter	University College London, UK
Nebojša Ikodinović	University of Belgrade, Serbia
Souhila Kaci	LIRMM, France
Sébastien Konieczny	CRIL, CNRS, France
Jérôme Lang	CNRS, LAMSADE, Universitè Paris-Dauphine, France

Fast Structure Learning for Deep Feedforward Networks via Tree Skeleton Expansion

Zhourong Chen(✉), Xiaopeng Li, Zhiliang Tian, and Nevin L. Zhang(✉)

The Hong Kong University of Science and Technology, Hong Kong, China
{zchenbb,xlibo,ztianac,lzhang}@cse.ust.hk

Abstract. Despite the popularity of deep learning, structure learning for deep models remains a relatively under-explored area. In contrast, structure learning has been studied extensively for probabilistic graphical models (PGMs). In particular, an efficient algorithm has been developed for learning a class of tree-structured PGMs called hierarchical latent tree models (HLTMs), where there is a layer of observed variables at the bottom and multiple layers of latent variables on top. In this paper, we propose a simple unsupervised method for learning the structures of feedforward neural networks (FNNs) based on HLTMs. The idea is to expand the connections in the tree skeletons from HLTMs and to use the resulting structures for FNNs. Our method is very fast and it yields deep structures of virtually the same quality as those produced by the very time-consuming grid search method.

Keywords: Fast structure learning · Feedforward neural networks

1 Introduction

Deep learning has achieved great successes in the past few years [10,15,17,22]. More and more researchers are now starting to investigate the possibility of learning structures for deep models instead of constructing them manually [3,6,24,31]. There are three main objectives in structure learning: improving model performance, reducing model size, and saving manual labor and/or computation time. Most previous methods focus on the first and second objectives. For example, the goal of constructive algorithms [16] and neural architecture search [31] is to find network structures which can achieve good performance for specific tasks. Network pruning [9,18], on the other hand, aims to learn models which contain fewer parameters but still achieve comparable performance compared with dense models.

In this paper, we focus on the third objective. In practice, people usually determine model structure by manual tuning or grid-search. This is time-consuming as there can be a large number of hyper-parameter combinations to consider. We propose a fast unsupervised structure learning method for neural

G. Kern-Isberner and Z. Ognjanović (Eds.): ECSQARU 2019, LNAI 11726, pp. 277–289, 2019.
https://doi.org/10.1007/978-3-030-29765-7_23

networks. Our method determines the bulk of a network automatically, while allowing minor adjustments. It also learns the sparse connectivity between adjacent layers.

Our work is carried out in the context of standard feedforward neural networks (FNNs). While convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are designed for spatial and sequential data respectively, standard FNNs are used for data that are neither spatial nor sequential. The structures of CNNs and RNNs are relatively more sophisticated than those of FNNs. For example, a neuron at a convolutional layer is connected only to neurons in a small receptive field at the level below. The underlying assumption is that neurons in a small spatial region tend to be strongly correlated in their activations. In contrast, a neuron in an FNN is connected to all neurons at the level below. We aim to learn sparse FNN structures where a neuron is connected to only a small number of strongly correlated neurons at the level below.

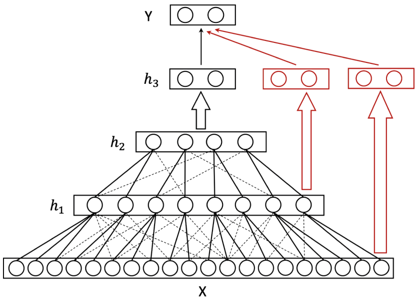

Fig. 1. Model structure of our Tree Skeleton Expansion Networks (TSE-Nets). The PGM core includes the bottom three layers $x - h_2$. The solid connections make up the skeleton and the dashed connections are added during the expansion phase. The black part of the model is called the Backbone, while the red part provides narrow skip-paths from the PGM core to the output layer.

Our work is built upon *hierarchical latent tree analysis* (HLTA) [5,20], an algorithm for learning tree-structured PGMs where there is a layer of observed variables at the bottom and multiple layers of latent variables on top. HLTA first partitions all the observed variables into groups such that the variables in each group are strongly correlated and the correlations can be better modelled using a single latent variable than using two. It then introduces a latent variable to explain the correlations among the variables in each group. After that it converts the latent variables into observed variables via data completion and repeats the process to produce a hierarchy.

To learn a sparse FNN structure, we assume data are generated from a PGM with multiple layers of latent variables and we try to approximately recover the structure of the generative model. To do so, we first run HLTA to obtain a tree model and use it as a *skeleton*. Then we expand it with additional edges to model salient probabilistic dependencies not captured by the skeleton. The result is a PGM structure and we call it a *PGM core*. To use the PGM core for classification, we further introduce a small number of neurons for each layer, and we connect them to all the units at the layers and all output units. This is to allow features from all layers to contribute to classification directly.

Figure 1 illustrates the result of our method. The PGM core includes the bottom three layers $x - h_2$. The solid connections make up the skeleton and the dashed connections are added during the expansion phase. The neurons at layer h_3 and the output units are added at the last step. The neurons at layer h_3 can be conceptually divided into two groups: those connected to the top layer of the PGM core and those connected to other layers. The PGM core, the first group at layer h_3 and the output units together form the *Backbone* of the model, while the second group at layer h_3 provide narrow *skip-paths* from low layers of the PGM core to the output layer. As the structure is obtained by expanding the connections of a tree skeleton, our model is called *Tree Skeleton Expansion Network* (TSE-Net).

2 Related Works

The primary goal in structure learning is to find a model with optimal or close-to-optimal generalization performance. Brute-force search is not feasible because the search space is large and evaluating each model is costly as it necessitates model training. Early works in the 1980's and 1990's have focused on what we call the *micro expansion* approach where one starts with a small network and gradually adds new neurons to the network until a stopping criterion is met [2,4,16]. The word "micro" is used here because at each step only one or a few neurons are added. This makes learning large model computationally difficult as reaching a large model would require many steps and model evaluation is needed at each step. In addition, those early methods typically do not produce layered structures that are commonly used nowadays. Recently, a *macro expansion* method [19] has been proposed where one starts from scratch and repeatedly add layers of hidden units until a threshold is met.

Other recent efforts have concentrated on what we call the *contraction* approach where one starts with a larger-than-necessary structure and reduces it to the desired size. Contraction can be done either by repeatedly pruning neurons and/or connections [9,18,26], or by using regularization to force some of the weights to zero [28]. From the perspective of structure learning, the contraction approach is not ideal because it requires a complex model as input. After all, a key motivation for a user to consider structure learning is to avoid building models manually.

A third approach is to explore the model space stochastically. One way is to place a prior over the space of all possible structures and carry out MCMC sampling to obtain a collection of models with high posterior probabilities [1]. Another way is to encode a model structure as a sequence of numbers, use a reinforcement learning meta model to explore the space of such sequences, learn a good meta policy from the sequences explored, and use the policy to generate model structures [31]. An obvious drawback of such *stochastic exploration* method is that they are computationally very expensive.

All the aforementioned methods learn model structures from supervised feedback. While useful, class labels contain far less information than model structures. As pointed out by [8], "The process of classification discards most of the information in the input and produces a single output (or a probability distribution over values of that single output)." In other words, there are rich information in data beyond class labels that we can make use of. As such, there are severe limitations if one relies only on supervised information to determine model structures. In this paper, we propose a novel structure learning method that makes use of unsupervised information in data. The method is called *skeleton expansion*. We first learn a tree-structured model based on correlations among variables, and then add a certain number of new units and connections to it in one shot. The method has two advantages: First, learning tree models is easier than learning non-tree models; Second, we need to train only one non-tree model, i.e., the final model.

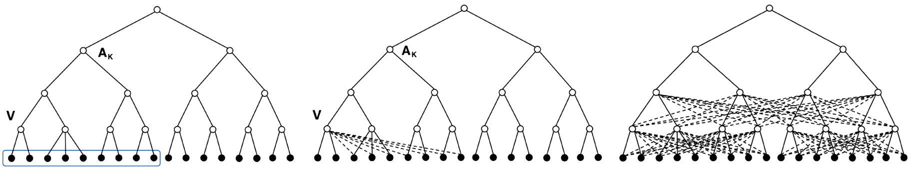

Fig. 2. Tree skeleton expansion. Left: A multi-layer tree skeleton is first learned. A_K is the ancestor of V which is $K = 2$ layers above V. Nodes in the blue circle are the descendants of A_K at the layer below V. Middle: New connections are then added to connect V to all the descendants of A_K at the layer below V. Right: Expansion is conducted on all the layers exception the top K layers. Read Sect. 4 for more details.

The skeleton expansion idea has been used in [6] to learn structures for restricted Boltzmann machines, which have only one hidden layer. This is the first time that the idea is applied to and tested on multi-layer feedforward networks.

3 Learning Tree Skeleton via HLTA

The first step of our method is to learn a tree-structured probabilistic graphical model $\mathcal{T}$ (an example $\mathcal{T}$ is shown in the left panel in Fig. 2). Let $\mathbf{X}$ be the set of observed variables at the bottom and $\mathbf{H}$ be the latent variables. Then $\mathcal{T}$ defines a joint distribution over all the variables:

$$P(\mathbf{X}, \mathbf{H}) = \prod_{v \in \{\mathbf{X}, \mathbf{H}\}} p(v | pa(v)),$$

where $pa(v)$ denotes the parent variable of v in $\mathcal{T}$. The distribution of $\mathbf{X}$ can be computed as:

$$P(\mathbf{X}) = \sum_{\mathbf{H}} P(\mathbf{X}, \mathbf{H}).$$

When learning the structure of $\mathcal{T}$, the objective is to maximize the BIC score [25] of $\mathcal{T}$ over data:

$$BIC(\mathcal{T}|D) = \log P(D|\theta^*) - \frac{d}{2}\log(N),$$

where θ^* is the maximum likelihood estimate of the parameters, d denotes the number of free parameters and D denotes the training data with N training samples. Guided by the BIC score, HLTA builds the structure in a layer-wise manner. It first partitions the observed variables into groups and learns a latent class model (LCM) [14] for each group. Let S denotes the set of observed variables which haven't been included into any variable groups. HLTA computes the mutual information for each pair of variables in S. Then it picks the pair with the highest mutual information and uses them as the seeds of a new variable group G. Other variables from S are then added to G one by one in descending order of their mutual information with variables already in G. Each time when a new variable is added into G, HLTA builds two models ($\mathcal{M}_1$ and $\mathcal{M}_2$). The two models are the best models with one single latent variable and two latent variables respectively, as shown in Fig. 3. HLTA computes the BIC scores of the two models and tests whether the following condition is met:

$$BIC(\mathcal{M}_2|D) - BIC(\mathcal{M}_1|D) \leq \delta, \tag{1}$$

where δ is a threshold which is always set at 3 in our experiments. When the condition is met, the two latent variable model $\mathcal{M}_2$ is not significantly better than the one latent variable model $\mathcal{M}_1$. Correlations among variables in G are still well modelled using a single latent variable. Then HLTA keeps on adding new variables to G. If the test fails, HLTA takes the subtree in $\mathcal{M}_2$ which doesn't contain the newly added variable and identifies the observed variables in it as a finalized variable group. The group are then removed from S. And the above process is repeated on S until all the variables are partitioned into disjoint groups (Fig. 4(b)).

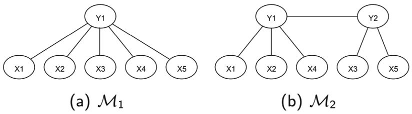

Fig. 3. Test whether five observed variables should be grouped together: (a) The best model with one latent variable. (b) The best model with two latent variables.

Next, HLTA introduces a latent variable for each variable group to explain the correlations among variables in the group. This results in a collection of latent class models, which are sometimes referred to as islands (Fig. 4(c), Left). To build the next layer, HLTA links up the islands and obtains what is called a flat model (Fig. 4(c), Right), and it turns the latent variables into observed variables by carrying out data completion within the flat model. Linking up the islands and carrying out data completion in the connected model is time-consuming. In this paper, we propose not to link up the islands. Instead, we carry out data completion in each individual island, which is crucial to speeding up the structure learning process. After converting the latent variables into observed variables $\mathbf{X}'$, the above process is repeated over $\mathbf{X}'$ to obtain another layer of latent variables (Fig. 4(d)). And this is repeated until there is only one new island (Fig. 4(e)). All the variables are then linked up as a multi-layer tree skeleton $\mathcal{T}$ with the last latent variable as the root (Fig. 4(f)).

4 Expanding Tree Skeleton to PGM Core

We have restricted the structure of $\mathcal{T}$ to be a tree, as parameter estimation in tree-structured PGMs is relatively efficient. However, this restriction in return also hurts the model's expressiveness. For example, in text analysis, the word *Apple* is highly correlated with both fruit words and technology words conceptually. But *Apple* is directly connected to only one latent variable in $\mathcal{T}$ and it is difficult for the single latent variable to express both the two concepts, which may cause severe underfitting. On the other hand, in standard FNNs, units at a layer are always fully connected to those at the previous layer, resulting in high connection redundancies.

In this paper, we aim to learn sparse connections between adjacent layers, such that they are neither as sparse as those in a tree, nor as dense as those in an FNN. To this end, the sparse connections should capture only the most important correlations among the observed variables. Thus we propose to use $\mathcal{T}$ as a structure skeleton and expand it to a denser structure $\mathcal{G}$ which we call the *PGM core*.

Our tree skeleton expansion method works as follows. For a node V at layer L, it finds the ancestor A_K of V that is K layers above V, and connects V to all descendants of A_K at layer $L-1$. We call K the up-looking parameter and set it to 2 in all our experiments. This means that each node is connected to all nodes in the "next generation" who descend from the same grandparent of the node (See Figure 2). We call it the "Grandparent expansion rule". This expansion phase is carried out over all the adjacent layers except the top K layers, as shown in Fig. 2 (Right). After the expansion phase, we take the bottom M layers and use the resulting sparse deep structure as the *PGM core* $\mathcal{G}$. M is a hyper-parameter that we need to determine in experiments.

5 Constructing Sparse FNNs from PGM Core

Our tree expansion method learns a multi-layer sparse structure $\mathcal{G}$ in an unsupervised manner. To utilize the resulting structure in a discriminative model, we convert each latent variable h in $\mathcal{G}$ to a hidden unit by defining:

$$h = o(\mathbf{W}'\mathbf{x} + b),$$

where $\mathbf{x}$ denotes a vector of the units directly connected to h at the layer below, $\mathbf{W}$ and b are connection weights and bias respectively, and o denotes a non-linear activation function, e.g. ReLU [7,23]. In this way, we convert $\mathcal{G}$ into a sparse multi-layer neural network. Next we discuss how we use it as a feature extractor in supervised learning tasks. Our model contains two parts, the *Backbone* and the *skip-paths.*

The Backbone. For a specific classification or regression task, we introduce a fully-connected layer on the top of $\mathcal{G}$, which we call the *feature layer*, followed by a output layer. As shown in Fig. 1, the feature layer serves as a feature "aggregator", aggregating the features extracted by $\mathcal{G}$ and feeding them to the output layer. We call the whole resulting module ($\mathcal{G}$, feature layer and output layer together) the Backbone, as it is supposed to be the major module of our model. The user needs to determine the number of units U at the feature layer. We set it to 100 in all our experiments.

The Skip-paths. As the structure of $\mathcal{G}$ is sparse and is learned to capture the strongest correlations in data, some weak but useful correlations may easily be missed. More importantly, different tasks may rely on different weak correlations and this cannot be taken into consideration during the unsupervised structure learning. To remedy this, we consider allowing the model to contain some narrow fully-connected paths to the feature layer such that they can capture those missed features. More specifically, as there are M layers of units in $\mathcal{G}$, we introduce $M-1$ more groups of units into the feature layer, with each group fully connected from

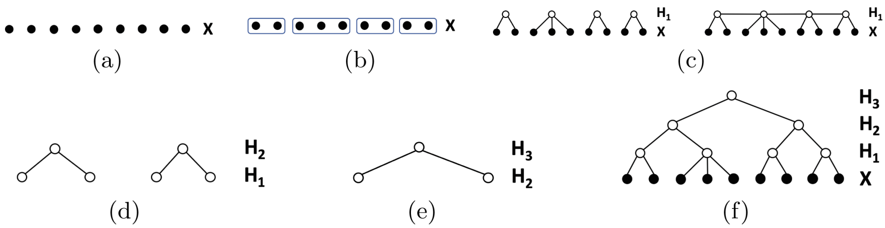

Fig. 4. The structure learning procedure for multi-layer tree skeleton. Black nodes represent observed variables while white nodes represent latent variables. (a) A set of observed variables X. (b) Partition the observed variables into groups. (c) Two options: (Left) Introduce a latent variable for each group. (Right) Introduce a latent variable for each group and link up the latent variables. The first option is much faster and is used in this paper. (d) Convert the layer-1 latent variables H_1 into observed variables and repeat the previous process on them. (e) Convert the layer-2 latent variables H_2 into observed variables and repeat (a)-(c) on them. (f) Stack the LCMs up to form a multi-layer tree skeleton.

a layer in $\mathcal{G}$ except the top one. In this way, each layer except the top one in $\mathcal{G}$ has both a sparse path (the Backbone) and a fully-connected path to the feature layer. The fully-connected paths are supposed to capture those minor features during parameter learning. These new paths are called *skip-paths*. Each group of units in the feature layer contains U units.

As shown in Fig. 1, the Backbone and the skip-paths together form our final model, named *Tree Skeleton Expansion Network* (TSE-Net). The model can then be trained like a normal neural network using back-propagation.

6 Discussion on Hyper-Parameters

There are four hyper-parameters, δ, K, M and U, in our method. The threshold δ is set at 3, which is determined from a threshold [5] suggested for Bayes factor. As for the up-looking parameter K and feature layer group size U, we suggest fixing $K = 2$ and $U = 100$ which work well on a range of datasets. The only hyper-parameter that the user needs to tune during training is the depth M of PGM core. The value depends heavily on the specific dataset. Tuning M allows the user to introduce minor adjustments to the model structure depending on the task and data.

7 Experiments

7.1 Datasets

We evaluate our method in 17 classification tasks. Table 1 gives a summary of the datasets. We choose 12 tasks of chemical compounds classification and 5 tasks of text classification. All the datasets are published by previous researchers.

Tox21 Challenge Dataset[1]. There are about 12,000 environmental chemical compounds in the dataset, each represented as its chemical structure. The tasks are to predict 12 different toxic effects for the chemical compounds [13,21]. We treat them as 12 binary classification tasks. We filter out sparse features which are present in fewer than 5% compounds, and rescale the remaining 1,644 features to zero mean and unit variance. The validation set is randomly sampled and removed from the original training set.

Text Classification Datasets[2]. We use 5 text classification datasets from [30]. After removing stop words, the top 10,000 frequent words in each dataset are selected as the vocabulary respectively and each document is represented as bag-of-words over the vocabulary. The validation set is randomly sampled and removed from the training samples.

[1] https://github.com/bioinf-jku/SNNs.
[2] https://github.com/zhangxiangxiao/Crepe.

Table 1. Statistics of all the datasets.

Dataset	Classes	Training samples	Validation samples	Test samples
Tox21	2	$\sim$ 9,000	500	$\sim$600
Yelp Review Full	5	640,000	10,000	50,000
DBPedia	14	549,990	10,010	70,000
Sogou News	5	440,000	10,000	60,000
Yahoo!Answer	10	1,390,000	10,000	60,000
AG's News	4	110,000	10,000	7,600

7.2 Experiment Setup

We compare our model TSE-Net with standard FNN. For fair comparison, we treat the number of units and number of layers as hyper-parameters of an FNN and optimize them via grid-search over all the defined combinations using validation data. Table 2 shows the space of network configurations considered, following the setup in [13]. In TSE-Net, the number of layers and number of units at each layer are determined by the algorithm.

We also compare our model with pruned FNN whose connections are sparse. We take the best FNN as the initial model and perform pruning as in [9]. As micro expansion and stochastic exploration methods are not learning layered FNNs and are computationally expensive, they are not included in comparison.

We use ReLUs [23] as the non-linear activation functions in all the networks. Dropout [11,27] with rate 0.5 is applied after each non-linear projection. We use Adam [12] as the network optimizer. Codes will be released after the paper is accepted.

7.3 Results

Training Time and Effective FLOP. The training time and effective FLOP (floating point operations) of TSE-Net, Backbone and FNN are reported in Table 3. The Total Time column shows the total time of structure learning/validation and network training in seconds. Effective FLOP is derived for the final model by calculating the number of non-zero weights, which can show us the computation saving when using the sparse model.

Table 2. Hyper-parameters for the structure of FNNs.

Hyper-parameter	Values considered
Number of units per layer	{512, 1024, 2048}
Number of hidden layers	{1,2,3,4}
Network shape	{Rectangle, Conic}

From the table we can see that, the training of TSE-Net and Backbone with unsupervised structure learning is significantly faster than that of FNN with grid-search, especially on large datasets. On the largest dataset, the training time ratio of TSE-Net w.r.t FNN is only 3.5%. And these differences can even be larger if we slightly increase the grid-search space. Moreover, our method is learning sparse models. The FLOP ratios of TSE-Net w.r.t FNN range from 7.01% to 37%, which means that our model can also save a significant part of computations in test time, given appropriate hardware support. In addition, our model is also containing much fewer parameters than the best FNN, which can save memory use in practice.

Classification. Classification results are reported in Table 4. All the experiments are run for three times and we report the average classification AUC scores/accuracies with standard deviations.

TSE-Nets vs FNNs. From the table we can see that, TSE-Net performs very close to the best FNN in 4 out of the 6 datasets, and achieves a 2.01% relative improvement on the Tox21 dataset. In our experiments, TSE-Net achieves better AUC scores than FNN in 9 out of the 12 tasks in the Tox21 dataset. It should be emphasized that, the structure of TSE-Net is trained in an unsupervised manner and it contains much fewer parameters than FNN, while the structure of FNN is manually optimized over the validation data. The results show that, the structure of TSE-Net successfully captures the crucial correlations in data and greatly reduces parameter number without significant performance loss.

It is worth noting that pure FNNs are not the state-of-the-art models for the tasks here. For example, [21] proposes an ensemble of FNNs, random forests and SVMs with expert knowledge for the Tox21 dataset. [13] tests different normalization techniques for FNNs on the Tox21 dataset. They both achieve an average AUC score around 0.846. Complicated RNNs [29] with attention also achieve better results than FNNs for the 5 text datasets. However, the goal of our paper is to learn sparse structure for FNNs, instead of proposing state-of-the-art methods for any specific tasks. Their methods are all much more complex and even task-specific, and hence it is not fair to include their results as comparison. Moreover, their methods can also be combined with ours to give better results.

Contribution of the Backbone. To validate our assumption that the Backbone in TSE-Net captures most of the crucial correlations in data and acts as a main part of the model, we remove the narrow skip-paths in TSE-Net and train the model to test its performance.

As we can see from the results, the Backbone path alone already achieves AUC scores or accuracies which are only slightly worse than those of TSE-Net. Note that the number of parameters in the Backbone is even much smaller than that of TSE-Net. The Backbone contains only 2%∼17% of the parameters in FNN. The results not only show the importance of the Backbone in TSE-Net, but also show that our structure learning for the Backbone path is effective.

Table 3. Time and sparsity. Total time contains time for structure learning/validation and network training in seconds. FLOP% column shows the FLOP ratio w.r.t FNN.

Task	TSE-Net (Ours)		Backbone		FNN	
	Total time	FLOP%	Total time	FLOP%	Total time	FLOP
Tox21 Average	128	17.25%	144	2.90%	154	1.64M
AG's News	1,107	7.01%	1,071	3.13%	11,099	28.88M
DBPedia	1,161	19.80%	1,327	8.98%	24,570	10.36M
Yelp Review Full	1,253	37.00%	1,332	16.06%	18,949	5.38M
Yahoo!Answer	1,315	36.70%	1,480	16.97%	37,475	5.39M
Sogou News	1,791	14.90%	1,806	6.38%	27,654	13.39M

Table 4. Classification results. All the experiments are run for three times and we report the average classification AUC scores/accuracies with standard deviations.

Task	TSE-Net (Ours)	Backbone	FNN	Pruned FNN
Tox21 Average	**0.8168**±0.0037	0.7856±0.0066	0.8010±0.0017	0.7998±0.0034
AG's News	91.49%±0.05%	91.54%±0.05%	**91.61**%±0.01%	91.49%±0.09%
DBPedia	**98.04**%±0.01%	97.74%±0.02%	97.99%±0.04%	97.95%±0.02%
Yelp Review Full	58.98%±0.09%	58.38%±0.07%	**59.13**%±0.14%	58.83%±0.01%
Yahoo!Answer	71.48%±0.12%	70.72%±0.02%	**71.84**%±0.07%	71.74%±0.05%
Sogou News	95.91%±0.01%	95.44%±0.06%	96.11%±0.06%	**96.20**%±0.06%

TSE-Nets vs Pruned FNNs. We also compare our method with a baseline method [9] for obtaining sparse FNNs. The pruning method provides regularization over the weights of a network. The regularization is even stronger than $l1/l2$ norm as it is producing many weights being exactly zeros. We start from the fully pretrained FNNs reported in Table 4, and prune the weak connections with the smallest absolute weight values. The pruned networks are then retrained again to compensate for the removed connections. After pruning, the number of remaining parameters in each FNN is the same as that in the corresponding TSE-Net for the same task. As shown in Table 4, TSE-Net and pruned FNN achieve pretty similar results. Note again that pruned FNN took much longer time than TSE-Net. Without any supervision or pre-training over connection weights, our unsupervised structure learning successfully identifies important connections and learns sparse structures. TSE-Net also achieves better interpretability as shown in https://arxiv.org/pdf/1803.06120.pdf.

8 Conclusions

It is important to the applications of deep learning to quickly learn a model structure appropriate for the problem at hand. A fast unsupervised structure learning method is proposed and investigated in this paper. In comparison with standard FNN, our model contains much fewer parameters and it takes much shorter time to learn. It also achieves comparable classification results in a range of tasks. Our method is also shown to learn models with better interpretability. In the future, we will generalize our method to networks like RNNs and CNNs.

Acknowledgments. Research on this article was supported by Hong Kong Research Grants Council under grants 16212516.

References

1. Adams, R.P., Wallach, H.M., Ghahramani, Z.: Learning the structure of deep sparse graphical models. In: AISTATS (2010)
2. Ash, T.: Dynamic node creation in backpropagation networks. Connection Sci. **1**(4), 365–375 (1989)
3. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. In: ICLR (2017)
4. Bello, M.G.: Enhanced training algorithms, and integrated training/architecture selection for multilayer perceptron networks. IEEE Trans. Neural Networks **3**, 864–875 (1992)
5. Chen, P., Zhang, N.L., Liu, T., Poon, L.K., Chen, Z., Khawar, F.: Latent tree models for hierarchical topic detection. Artif. Intell. **250**, 105–124 (2017)
6. Chen, Z., Zhang, N.L., Yeung, D.Y., Chen, P.: Sparse Boltzmann machines with structure learning as applied to text analysis. In: AAAI (2017)
7. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: AISTATS (2011)
8. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
9. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: NIPS (2015)
10. Hinton, G.E., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Sig. Process. Mag. **29**(6), 82–97 (2012)
11. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012)
12. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
13. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. In: NIPS (2017)
14. Knott, M., Bartholomew, D.J.: Latent Variable Models and Factor Analysis (1999)
15. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS (2012)
16. Kwok, T.Y., Yeung, D.Y.: Constructive algorithms for structure learning in feedforward neural networks for regression problems. IEEE Trans. Neural Networks **8**(3), 630–645 (1997)
17. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)

18. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: ICLR (2017)
19. Liu, J., Gong, M., Miao, Q., Wang, X., Li, H.: Structure learning for deep neural networks based on multiobjective optimization. IEEE Trans. Neural Networks Learn. Syst. **29**, 2450–2463 (2017)
20. Liu, T., Zhang, N.L., Chen, P.: Hierarchical latent tree analysis for topic detection. In: Calders, T., Esposito, F., Hüllermeier, E., Meo, R. (eds.) ECML PKDD 2014. LNCS (LNAI), vol. 8725, pp. 256–272. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44851-9_17
21. Mayr, A., Klambauer, G., Unterthiner, T., Hochreiter, S.: Deeptox: toxicity prediction using deep learning. Front. Environ. Sci. **3**, 80 (2016)
22. Mikolov, T., Deoras, A., Povey, D., Burget, L., Černockỳ, J.: Strategies for training large scale neural network language models. In: IEEE Workshop on Automatic Speech Recognition and Understanding, pp. 196–201 (2011)
23. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: ICML (2010)
24. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Le, Q., Kurakin, A.: Large-scale evolution of image classifiers. In: ICML (2017)
25. Schwarz, G., et al.: Estimating the dimension of a model. Ann. Stat. **6**(2), 461–464 (1978)
26. Srinivas, S., Babu, R.V.: Data-free parameter pruning for deep neural networks. In: Proceedings of the British Machine Vision Conference (2015)
27. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. JMLR **15**(1), 1929–1958 (2014)
28. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: NIPS (2016)
29. Yang, Z., Yang, D., Dyer, C., He, X., Smola, A.J., Hovy, E.H.: Hierarchical attention networks for document classification. In: HLT-NAACL (2016)
30. Zhang, X., Zhao, J., LeCun, Y.: Character-level convolutional networks for text classification. In: NIPS (2015)
31. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: ICLR (2017)

So, in order to evaluate $\left[\overline{T}f\right](x)$, one must solve an optimisation problem over the set $\mathscr{T}_x$ containing the x-rows of the elements of $\mathscr{T}$. In many practical cases, the set $\mathscr{T}_x$ is closed and convex and therefore, evaluating $\left[\overline{T}f\right](x)$ is relatively straightforward: for instance, if $\mathscr{T}_x$ is described by a finite number of (in)equality constraints, then this problem reduces to a simple linear programming task, which can be solved by standard techniques. We will also make use of the conjugate *lower transition operator* $\underline{T}$, defined by $[\underline{T}\,f](x) := -[\overline{T}(-f)](x)$ for all $x \in \mathscr{X}$ and all $f \in \mathscr{L}(\mathscr{X})$. Results about upper transition operators translate to results about lower transition operators through this relation; we will focus on the former in the following discussion.

Now, the operator $\overline{T}$ can be used for computing upper expectations in much the same way as transition matrices are used for computing expectations with respect to precise Markov chains: for any $n \in \mathbb{N}$, any finitary function $f(X_{n+1})$ and any $x_{1:n} \in \mathscr{X}^n$ it holds that

$$\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_{n+1})|X_{1:n} = x_{1:n}) = \left[\overline{T}f\right](x_n)\,. \tag{1}$$

Observe that the right-hand side in this expression does not depend on the history $x_{1:n-1}$; this can be interpreted as saying that the model satisfies an *imprecise Markov property*, which explains why we call our model an "imprecise Markov chain". Moreover, a slightly more general property holds that will be useful later on:

Proposition 1. *Consider the imprecise Markov chain $\mathscr{P}_{\mathcal{M},\mathscr{T}}$. For any $m, n \in \mathbb{N}$ such that $m \leq n$, any function $f \in \mathscr{L}(\mathscr{X}^{n-m+1})$ and any $x_{1:m-1} \in \mathscr{X}^{m-1}$ and $y \in \mathscr{X}$, we have that*

$$\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}\left(f(X_{m:n})\middle|X_{1:m-1} = x_{1:m-1}, X_m = y\right) = \overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}\left(f(X_{1:n-m+1})\middle|X_1 = y\right).$$

Finally, we remark that, for any $m, n \in \mathbb{N}$ such that $m \leq n$, a conditional upper expectation $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}\left(f\middle|X_{m:n}\right)$ is itself a (finitary) function depending on the states $X_{m:n}$. Using this observation, we can now introduce the *law of iterated upper expectations*, which will form the basis of the algorithms developed in the following sections:

Theorem 1. *Consider the imprecise Markov chain $\mathscr{P}_{\mathcal{M},\mathscr{T}}$. For all $m \in \mathbb{N}_0$, all $k \in \mathbb{N}$ and all $f \in \mathscr{L}_{\mathrm{fin}}(\Omega)$, we have that*

$$\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}\left(f\middle|X_{1:m}\right) = \overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}\left(\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}\left(f\middle|X_{1:m+k}\right)\middle|X_{1:m}\right).$$

4 A Recursive Inference Algorithm

In principle, for any function $f \in \mathscr{L}(\mathscr{X}^n)$ with $n \in \mathbb{N}$, the upper expectations of $f(X_{1:n})$ can be obtained by maximising $\mathrm{E}_{\mathrm{P}}(f(X_{1:n}))$ over the set $\mathscr{P}_{\mathcal{M},\mathscr{T}}$ of all precise models P that are compatible with $\mathcal{M}$ and $\mathscr{T}$. Since this will almost always be infeasible if n is large, we usually apply the law of iterated upper expectations in combination with the Markov property in order to divide the optimisation problem into multiple smaller ones. Indeed, because of Theorem 1, we have that

$$\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_{1:n})) = \overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}\Big(\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_{1:n})|X_{1:n-1})\Big).$$

Using Eq. (1), one can easily show that $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_{1:n})|X_{1:n-1})$ can be computed by evaluating $[\overline{T}f(x_{1:n-1}\cdot)](x_{n-1})$ for all $x_{1:n-1} \in \mathscr{X}^{n-1}$. Here, $f(x_{1:n-1}\cdot)$ is the function in $\mathscr{L}(\mathscr{X})$ that takes the value $f(x_{1:n})$ on $x_n \in \mathscr{X}$. This accounts for $|\mathscr{X}|^{n-1}$ optimisation problems to be solved. With the acquired function $f'(X_{1:n-1}) := \overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_{1:n})|X_{1:n-1})$, we can then compute the upper expectation $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f'(X_{1:n-1})|X_{1:n-2})$ in a similar way, by solving $|\mathscr{X}|^{n-2}$ optimisation problems. Continuing in this way, we end up with a function that only depends on X_1 and for which the expectation needs to be maximised over the initial models in $\mathcal{M}$. Hence, in total, $\sum_{i=0}^{n-1}|\mathscr{X}|^i$ optimisation problems need to be solved in order to obtain $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_{1:n}))$. Although these optimisation problems are relatively simple and therefore feasible to solve individually, the total number of required iterations is still exponential in n, therefore making the computation of $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_{1:n}))$ intractable when n is large.

In many cases, however, $f(X_{1:n})$ can be recursively decomposed in a specific way allowing for a much more efficient computational scheme to be employed; see Theorem 2 further on. Before we present this scheme in full generality, let us first provide some intuition about its basic working principle.

So assume we are interested in $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_{1:n}))$, which, according to Theorem 1, can be obtained by maximising $\mathrm{E}_{\mathrm{P}}(\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_{1:n})|X_1))$ over $\mathrm{P}(X_1) \in \mathcal{M}$. The problem then reduces to the question of how to compute $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_{1:n})|X_1)$ efficiently. Suppose now that $f(X_{1:n})$ takes the following form:

$$f(X_{1:n}) = g(X_1) + h(X_1)\tau(X_{2:n}), \tag{2}$$

for some $g, h \in \mathscr{L}(\mathscr{X})$ and some $\tau \in \mathscr{L}(\mathscr{X}^{n-1})$. Then, because $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}$ is a supremum over linear expectations, we find that

$$\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_{1:n})|X_1) = g(X_1) + h(X_1)\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau(X_{2:n})|X_1),$$

where, for the sake of simplicity, we assumed that h does not take negative values. Then, by appropriately combining Proposition 1 with Theorem 1, one can express $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau(X_{2:n})|X_1)$ in terms of $\overline{\Upsilon}\colon \mathscr{X} \to \mathbb{R}$, defined by

$$\overline{\Upsilon}(x) := \overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau(X_{1:n-1})|X_1 = x) \text{ for all } x \in \mathscr{X}.$$

In particular, we find that

$$\begin{aligned}\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau(X_{2:n})|X_1) = \overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}\big(\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau(X_{2:n})|X_{1:2})|X_1\big) &= \overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}\big(\overline{\Upsilon}(X_2)|X_1\big)\\ &= [\overline{T}\,\overline{\Upsilon}](X_1),\end{aligned}$$

where the equalities follow from Theorem 1, Proposition 1 and Eq. (1), respectively. So $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_{1:n})|X_1)$ can be obtained from $\overline{\Upsilon}$ by solving a single optimisation problem, followed by a pointwise multiplication and summation.

Now, by repeating the structural assessment (2) in a recursive way, we can generate a whole class of functions for which the upper expectations can be computed using the principle illustrated above. We start with a function $\tau_1(X_1)$, with $\tau_1 \in \mathscr{L}(\mathscr{X})$, that only depends on the initial state. The upper expectation $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_1(X_1)|X_1)$ is then trivially equal to $\tau_1(X_1)$. Next, consider $\tau_2(X_{1:2}) = g_1(X_1) + h_1(X_1)\tau_1(X_2)$ for some g_1, h_1 in $\mathscr{L}(\mathscr{X})$. $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_2(X_{1:2})|X_1)$ is then given by $g_1(X_1) + h_1(X_1)[\overline{T}\,\overline{\Upsilon}_1](X_1)$, where we let $\overline{\Upsilon}_1(x) := \overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_1(X_1)|X_1 = x) = \tau_1(x)$ for all $x \in \mathscr{X}$ and where we (again) neglect the subtlety that h_1 can take negative values. Continuing in this way, step by step considering new functions constructed by multiplication and summation with functions that depend on an additional time instance, and no longer ignoring the fact that the functions involved can take negative values, we end up with the following result.

Theorem 2. *Consider any imprecise Markov chain $\mathscr{P}_{\mathcal{M},\mathscr{T}}$ and two sequences of functions $\{g_n\}_{n\in\mathbb{N}_0}$ and $\{h_n\}_{n\in\mathbb{N}}$ in $\mathscr{L}(\mathscr{X})$. Define $\tau_1(x_1) := g_0(x_1)$ for all $x_1 \in \mathscr{X}$, and for all $n \in \mathbb{N}$, let*

$$\tau_{n+1}(x_{1:n+1}) := h_n(x_1)\tau_n(x_{2:n+1}) + g_n(x_1) \text{ for all } x_{1:n+1} \in \mathscr{X}^{n+1}.$$

If we write $\{\overline{\Upsilon}_n\}_{n\in\mathbb{N}}$ and $\{\underline{\Upsilon}_n\}_{n\in\mathbb{N}}$ to denote the sequences of functions in $\mathscr{L}(\mathscr{X})$ that are respectively defined by $\overline{\Upsilon}_n(x) := \overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_n(X_{1:n})|X_1 = x)$ and $\underline{\Upsilon}_n(x) := \underline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_n(X_{1:n})|X_1 = x)$ for all $x \in \mathscr{X}$ and all $n \in \mathbb{N}$, then $\{\overline{\Upsilon}_n\}_{n\in\mathbb{N}}$ and $\{\underline{\Upsilon}_n\}_{n\in\mathbb{N}}$ satisfy the following recursive expressions:

$$\begin{cases} \overline{\Upsilon}_1 = \underline{\Upsilon}_1 = g_0; \\ \overline{\Upsilon}_{n+1} = h_n \mathbb{I}_{h_n \geq 0}[\overline{T}\,\overline{\Upsilon}_n] + h_n \mathbb{I}_{h_n < 0}[\underline{T}\,\underline{\Upsilon}_n] + g_n \text{ for all } n \in \mathbb{N}; \\ \underline{\Upsilon}_{n+1} = h_n \mathbb{I}_{h_n \geq 0}[\underline{T}\,\underline{\Upsilon}_n] + h_n \mathbb{I}_{h_n < 0}[\overline{T}\,\overline{\Upsilon}_n] + g_n \text{ for all } n \in \mathbb{N}. \end{cases}$$

Here, we used $\mathbb{I}_{h_n \geq 0} \in \mathscr{L}(\mathscr{X})$ to denote the indicator of $\{x \in \mathscr{X} : h_n(x) \geq 0\}$, and similarly for $\mathbb{I}_{h_n < 0} \in \mathscr{L}(\mathscr{X})$. Note that, because we now need to evaluate both $\overline{T}$ and $\underline{T}$ for every iteration, we will in general need to solve $2(n-1)|\mathscr{X}|$ optimisation problems to obtain $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_n(X_{1:n})|X_1)$ and $\underline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_n(X_{1:n})|X_1)$ for some $n \in \mathbb{N}$. In order to obtain the unconditional inferences $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_n(X_{1:n}))$ and $\underline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_n(X_{1:n}))$, it then suffices to respectively maximise and minimise the expectations of $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_n(X_{1:n})|X_1)$ and $\underline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_n(X_{1:n})|X_1)$ over all initial models in $\mathcal{M}$.

5 Special Cases

To illustrate the practical relevance of our method, we now discuss a number of important inferences that fall within its scope. As already mentioned in the introduction section, in some of these cases, our method simplifies to a computational scheme that was already developed earlier in a more specific context. The strength of our present contribution, therefore, lays in its unifying character and the level of generality to which it extends.

Functions that depend on a single time instant. As a first, very simple inference we can consider the upper and lower expectation of a function $f(X_n)$, for some $f \in \mathscr{L}(\mathscr{X})$ and $n \in \mathbb{N}$, conditional on the initial state. The expressions for these inferences are given by $\overline{T}^{n-1}f$ and $\underline{T}^{n-1}f$, respectively [5]. For instance, for any $x \in \mathscr{X}$, $\underline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f(X_5))|X_1 = x) = [\underline{T}^4 f](x)$. These expressions can also easily be obtained from Theorem 2, by setting $g_0 := f$ and, for all $k \in \{1, \cdots, n-1\}$, $g_k := 0$ and $h_k := 1$.

Sums of functions. One can also use our method to compute upper and lower expectations of sums $\sum_{k=1}^{n} f_k(X_k)$ of functions $f_k \in \mathscr{L}(\mathscr{X})$. Then we would have to set $g_0 := f_n$ and, for all $k \in \{1, \cdots, n-1\}$, $g_k := f_{n-k}$ and $h_k := 1$. Although we allow the functions f_k to depend on k, it is worth noting that, if we set them all equal to the same function f, our method can also be employed to compute the upper and lower expectation of the time average $^1/_n \sum_{k=1}^{n} f(X_k)$ of f over the time interval n. The subtlety of the constant factor $^1/_n$ does not raise a problem here, because upper and lower expectations are homogeneous with respect to non-negative scaling.

Product of functions. Another interesting class of inferences are those that can be represented by a product $\prod_{k=1}^{n} f_k(X_k)$ of functions $f_k \in \mathscr{L}(\mathscr{X})$. To compute upper and lower expectations of such functions, it suffices to set $g_0 := f_n$ and, for all $k \in \{1, \cdots, n-1\}$, $g_k := 0$ and $h_k := f_{n-k}$. A typical example of an inference than can be described in this way is the probability that the state will be in a set $A \subseteq \mathscr{X}$ during a certain time interval. For instance, the upper expectation of the function $\mathbb{I}_A(X_1)\mathbb{I}_A(X_2)$ gives us a tight upper bound on the probability that the state will be in A during the first two time instances.

Hitting probabilities. The hitting probability of some set $A \subseteq \mathscr{X}$ over a finite time interval n is the probability that the state X_k will be in A somewhere within the first n time instances. The upper and lower bounds on such a hitting probability are equal to the upper and lower expectation of the function $f(X_{1:n}) := \mathbb{I}_{A'_n} \in \mathscr{L}(\Omega)$, where $A'_n := \{\omega \in \Omega \colon (\exists k \leq n)\, \omega_k \in A\}$. Note that $f(X_{1:n})$ can be decomposed in the following way:

$$f(X_{1:n}) = \mathbb{I}_A(X_1) + \mathbb{I}_A(X_2)\mathbb{I}_{A^c}(X_1) + \cdots + \mathbb{I}_A(X_n) \prod_{k=1}^{n-1} \mathbb{I}_{A^c}(X_k)$$

Hence, these inferences can be obtained using Theorem 2 if we let $g_0 := \mathbb{I}_A$ and, for all $k \in \{1, \cdots, n-1\}$, $g_k := \mathbb{I}_A$ and $f_k := \mathbb{I}_{A^c}$. Additionally, one could also be interested in the probability that the state X_k will *ever* be in A. Upper and lower bounds on this probability are given by the upper and lower expectation of the function $f := \mathbb{I}_{A'} \in \mathscr{L}(\Omega)$ where $A' := \{\omega \in \Omega \colon (\exists k \in \mathbb{N})\, \omega_k \in A\}$. Since the function f is non-finitary, we are unable to apply our method in a direct way. However, it is shown in [8, Proposition 16] that, if the set $\mathscr{T}$ is convex and closed, the upper and lower bounds on the hitting probability over a finite time interval converge to the upper and lower bounds on the hitting probability over an infinite time interval, therefore allowing us to approximate these inferences by choosing n sufficiently large.

Hitting times. The *hitting time* of some set $A \subseteq \mathscr{X}$ is defined as the time τ until the state is in A for the first time; so $\tau(\omega) := \inf\{k \in \mathbb{N}_0 : \omega_k \in A\}$ for all $\omega \in \Omega$. Once more, the function τ is non-finitary, necessitating an indirect approach to the computation of its upper and lower expectation. This can be done in a similar way as we did for the case of hitting probabilities, now considering the finitary functions $\tau_n(X_{1:n})$, where $\tau_n \in \mathscr{L}(\mathscr{X}^n)$ is defined by $\tau_n(x_{1:n}) := \inf\{k \in \mathbb{N} : x_k \in A\}$ if $\{k \in \mathbb{N} : x_k \in A\}$ is non-empty, and $\tau_n(x_{1:n}) := n+1$ otherwise, for all $n \in \mathbb{N}$ and all $x_{1:n} \in \mathscr{X}^n$. These functions correspond to choosing $g_0 := \mathbb{I}_{A^c}$ and, for all $k \in \{1, \cdots, n-1\}$, $g_k := \mathbb{I}_{A^c}$ and $f_k := \mathbb{I}_{A^c}$. If the set $\mathscr{T}$ is convex and closed, the upper and lower expectations of these functions for large n will then approximate those of the non-finitary hitting time [8, Proposition 10].

6 Discussion

The main contribution of this paper is a single, unified method to efficiently compute a wide variety of inferences for imprecise Markov chains; see Theorem 2. The set of functions describing these inferences is however restricted to the finitary type, and therefore a general approach for inferences characterised by non-finitary functions is still lacking. In some cases, however, as we already mentioned in our discussion of hitting probabilities and hitting times, this issue can be addressed by relying on a continuity argument.

Indeed, consider any function $f = \lim_{n\to+\infty} \tau_n(X_{1:n})$ that is the pointwise limit of a sequence $\{\tau_n(X_{1:n})\}_{n\in\mathbb{N}}$ of finitary functions, defined recursively as in Theorem 2. If $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}$ is continuous with respect to $\{\tau_n(X_{1:n})\}_{n\in\mathbb{N}}$, meaning that $\lim_{n\to+\infty} \overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_n(X_{1:n})) = \overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f)$, the inference $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f)$ can then be approximated by $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_n(X_{1:n}))$ for sufficiently large n. Since we can recursively compute $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(\tau_n(X_{1:n}))$ for any $n \in \mathbb{N}$ using the methods discussed at the end of Sect. 4, this yields an efficient way of approximating $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f)$. A completely analogous argument can be used for the lower expectation $\underline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}(f)$. This begs the question whether the upper and lower expectations $\overline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}$ and $\underline{\mathrm{E}}_{\mathcal{M},\mathscr{T}}$ satisfy the appropriate continuity properties for this to work.

Unfortunately, results about the continuity properties of these operators are rather scarce —especially compared to their precise counterparts—and depend on the formalism that is adopted. In this paper, for didactical reasons, we have considered one formalism: we have introduced imprecise Markov chains as being *sets* of "precise" models that are in a specific sense compatible with the given set $\mathscr{T}$. It is however important to realise that there is also an entirely different formalisation of imprecise Markov chains that is instead based on the game-theoretic probability framework that was popularised by Shafer and Vovk; we refer to [10,11] for details. It is well known that the inferences produced under these two different frameworks agree for finitary functions [3,9], so the method described by Theorem 2 is also applicable when working in a game-theoretic framework. The continuity properties of the game-theoretic upper and lower expectations, however, are not necessarily the same as those of the measure-theoretic operators that we considered here. So far, the continuity properties of game-theoretic upper and lower expectations are better understood [10–12], making these operators more suitable if we plan to employ the continuity argument above.

Acknowledgments. The work in this paper was partially supported by H2020-MSCA-ITN-2016 UTOPIAE, grant agreement 722734.

References

1. De Bock, J.: Credal networks under epistemic irrelevance. Int. J. Approximate Reasoning **85**, 107–138 (2017)
2. De Bock, J.: Credal networks under epistemic irrelevance: theory and algorithms. Ph.D. thesis, Ghent University (2015)
3. de Cooman, G., Hermans, F.: Imprecise probability trees: bridging two theories of imprecise probability. Artif. Intell. **172**(11), 1400–1427 (2008)
4. de Cooman, G., Hermans, F., Antonucci, A., Zaffalon, M.: Epistemic irrelevance in credal nets: the case of imprecise Markov trees. Int. J. Approximate Reasoning **51**(9), 1029–1052 (2010)
5. de Cooman, G., Hermans, F., Quaeghebeur, E.: Imprecise Markov chains and their limit behaviour. Probab. Eng. Inf. Sci. **23**(4), 597–635 (2009)
6. Hermans, F., Škulj, D.: Stochastic processes. In: Augustin, T., Coolen, F.P.A., De Cooman, G., Troffaes, M.C.M. (eds.) Introduction to Imprecise Probabilities. Wiley (2014). Chapter 11
7. Kallenberg, O.: Foundations of Modern Probability. Springer Science & Business Media, New York (2006)
8. Krak, T'Joens, N., De Bock, J.: Hitting times and probabilities for imprecise Markov chains (2019). Accepted for publication in the conference proceedings of ISIPTA 2019. A preprint can be found at arXiv:1905.08781
9. Lopatatzidis, S.: Robust Modelling and Optimisation in Stochastic Processes using Imprecise Probabilities, with an Application to Queueing Theory. Ph.D. thesis, Ghent University (2017)
10. Shafer, G., Vovk, V.: Probability and Finance: It's Only a Game!. Wiley, New York (2001)
11. T'Joens, N., De Bock, J., de Cooman, G.: In search of a global belief model for discrete-time uncertain processes (2019). Accepted for publication in the conference proceedings of ISIPTA 2019
12. T'Joens, N., De Bock, J., de Cooman, G.: Continuity properties of game-theoretic upper expectations. arXiv:1902.09406 (2019)
13. T'Joens, N., Krak, T., De Bock, J., de Cooman, G.: A Recursive Algorithm for Computing Inferences in Imprecise Markov Chains. arXiv:1905.12968 (2019)

Uncertain Reasoning for Applications