The Journal of Open Source Software

TensorInference: A Julia package for tensor-based probabilistic inference

Martin Roa-Villescas [1*] and **Jin-Guo Liu** [2*]

1 Eindhoven University of Technology **2** Hong Kong University of Science and Technology (Guangzhou)
***** These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

`TensorInference.jl` is a Julia (Bezanson et al., 2017) library designed for performing probabilistic inference in discrete graphical models. It leverages the recent explosion of advances in the field of tensor networks (Orús, 2019) to provide high-performance solutions for common inference tasks. These tasks include calculating: 1) the partition function or probability of evidence, 2) the marginal probability distribution over each variable given evidence, 3) the most likely assignment to all variables given evidence, 4) the most likely assignment to the query variables after marginalizing out the remaining variables, and 5) samples generated from the variable probability distribution given evidence. The infrastructure based on tensor networks Liu et al. (2022) allows users to define the contraction ordering method, which is known to have a significant impact on the computational performance Pan & Zhang (2021) of these algorithms (Orús, 2014). A predefined set of state-of-the-art contraction ordering methods is made available to users. These methods include the *recursive multi-tensor contraction method* (`TreeSA`) (Kalachev et al., 2022), the *hyper-optimized tensor network contraction method* (`KaHyParBipartite`) (Gray & Kourtis, 2021) and its simulated annealing variant (`SABipartite`), and a *greedy-based memory minimization method* (`GreedyMethod`). Finally, `TensorInference.jl` harnesses the latest developments in computational technology, including a highly optimized set of BLAS (Blackford et al., 2002) routines and GPU technology.

Statement of need

A major challenge in developing intelligent systems is the ability to reason under uncertainty, a challenge that appears in many real-world problems across various domains, including artificial intelligence, medical diagnosis, computer vision, computational biology, and natural language processing. Reasoning under uncertainty involves calculating the probabilities of relevant variables while taking into account any information that is acquired. This process, which can be thought of as drawing global insights from local observations, is known as *probabilistic inference*.

Probabilistic graphical models (PGMs) provide a unified framework to perform probabilistic inference. These models use graphs to represent the joint probability distribution of complex systems concisely by exploiting the conditional independence between variables in the model. Additionally, they form the foundation for various algorithms that enable efficient probabilistic inference.

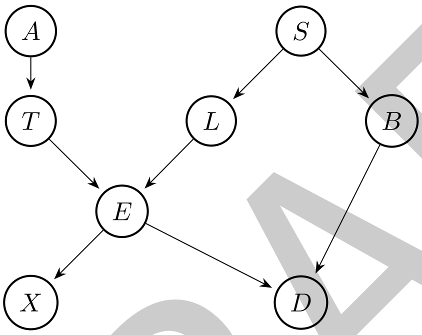
However, even with the representational aid of PGMs, performing probabilistic inference remains an intractable endeavor on many real-world models. The reason is that performing probabilistic inference involves complex combinatorial optimization problems in very high dimensional spaces. To tackle these challenges, more efficient and scalable inference algorithms are needed.

As an attempt to tackle the aforementioned challenges, we present TensorInference.jl, a Julia package for probabilistic inference that combines the representational capabilities of PGMs with the computational power of tensor networks. By harnessing the best of both worlds, TensorInference.jl aims to enhance the performance of probabilistic inference, thereby expanding the tractability spectrum of exact inference for more complex, real-world models.

In contrast with the JunctionTrees.jl package (Roa-Villescas et al., 2022, 2023), which utilizes a tensor-based backend to optimize the computation of individual sum-product messages within the context of the junction tree algorithm (JTA) (Jensen et al., 1990; Lauritzen & Spiegelhalter, 1988), TensorInference.jl adopts a holistic approach. This approach subsumes the JTA in its entirety, greatly simplifying the complexity of the algorithm and opening new doors for optimization opportunities.

Usage example

The graph below corresponds to the *ASIA network* (Lauritzen & Spiegelhalter, 1988), a simple Bayesian network (Pearl, 1985) used extensively in educational settings.



Random variable	Meaning
A	Recent trip to Asia
T	Patient has tuberculosis
S	Patient is a smoker
L	Patient has lung cancer
B	Patient has bronchitis
E	Patient has T and/or L
X	Chest X-Ray is positive
D	Patient has dyspnoea

Figure 1: The ASIA network: a simplified example of a Bayesian network from the context of medical diagnosis (Lauritzen & Spiegelhalter, 1988). It describes the probabilistic relationships between different random variables which correspond to possible diseases, symptoms, risk factors and test results.

We now demonstrate how to use TensorInference.jl for conducting a variety of inference tasks on this toy example.

```julia
# Import the TensorInference package, which provides the functionality needed
# for working with tensor networks and probabilistic graphical models.
using TensorInference

# Load the ASIA network model from the `asia.uai` file located in the examples
# directory. Refer to the documentation of this package for a description of the
# format of this file.
instance = read_instance(pkgdir(TensorInference, "examples", "asia", "asia.uai"))

# Create a tensor network representation of the loaded model.
# The variable 7 is the variable of interest, which will be retained in the output.
tn = TensorNetworkModel(instance; openvars=[7])

# Calculate the partition function for each assignment of variable 7.
probability(tn)

# Calculate the marginal probabilities of each random variable in the model.
```

```julia
marginals(tn)

# Retrieve the variables associated with the tensor network model.
get_vars(tn)

# Assume that the "X-ray" result (variable 7) is positive.
# Since setting an evidence may affect the contraction order of the tensor
# network, recompute it.
tn = TensorNetworkModel(instance; evidence=Dict(7 => 0))

# Calculate the maximum log-probability among all configurations.
maximum_logp(tn)

# Generate 10 samples from the probability distribution represented by the
# model.
sample(tn, 10)

# Retrieve both the maximum log-probability and the most probable
# configuration. In this configuration, the most likely outcomes are that the
# patient smokes (variable 3) and has lung cancer (variable 4).
logp, cfg = most_probable_config(tn)

# Compute the most probable values of certain variables (e.g., 4 and 7) while
# marginalizing over others. This is known as Maximum a Posteriori (MAP)
# estimation.
mmap = MMAPModel(instance, queryvars=[4, 7])

# Get the most probable configurations for variables 4 and 7.
most_probable_config(mmap)

# Compute the total log-probability of having lung cancer. The results suggest
# that the probability is roughly half.
log_probability(mmap, [1, 0]), log_probability(mmap, [0, 0])
```

Acknowledgments

This work is partially funded by the Netherlands Organization for Scientific Research. The authors want to thank Madelyn Cain for helpful advice.

References

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. (2017). Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Blackford, L. S., Petitet, A., Pozo, R., Remington, K., Whaley, R. C., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., & others. (2002). An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, *28*(2), 135–151.

Gray, J., & Kourtis, S. (2021). Hyper-optimized tensor network contraction. *Quantum*, *5*, 410. https://doi.org/10.22331/q-2021-03-15-410

Jensen, F. V., Lauritzen, S. L., & Olesen, K. G. (1990). Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, *4*, 269–282.

Kalachev, G., Panteleev, P., & Yung, M.-H. (2022). *Multi-tensor contraction for XEB verification of quantum circuits*. https://arxiv.org/abs/2108.05665

Lauritzen, S. L., & Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, *50*(2), 157–194.

Liu, J. G., Gao, X., Cain, M., Lukin, M. D., & Wang, S.-T. (2022). *Computing solution space properties of combinatorial optimization problems via generic tensor networks*. arXiv. https://doi.org/10.48550/ARXIV.2205.03718

Markov, I. L., & Shi, Y. (2008). Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing*, *38*(3), 963–981. https://doi.org/10.1137/050644756

Orús, R. (2014). A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, *349*, 117–158. https://doi.org/10.1016/j.aop.2014.06.013

Orús, R. (2019). Tensor networks for complex quantum systems. *Nature Reviews Physics*, *1*(9), 538–550.

Pan, F., & Zhang, P. (2021). *Simulating the sycamore quantum supremacy circuits*. https://arxiv.org/abs/2103.03074

Pearl, J. (1985). Bayesian networks: A model of self-activated memory for evidential reasoning. *Proc. Of Cognitive Science Society (CSS-7)*.

Roa-Villescas, M., Liu, J. G., Wijnings, P. W. A., Stuijk, S., & Corporaal, H. (2023). Scaling probabilistic inference through message contraction optimization. *2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE)*.

Roa-Villescas, M., Wijnings, P. W. A., Stuijk, S., & Corporaal, H. (2022). Partial evaluation in junction trees. *2022 25th Euromicro Conference on Digital System Design (DSD)*, 429–437. https://doi.org/10.1109/DSD57027.2022.00064

Robeva, E., & Seigal, A. (2019). Duality of graphical models and tensor networks. *Information and Inference: A Journal of the IMA*, *8*(2), 273–288. https://doi.org/10.1093/imaiai/iay009