Learning Sum-Product Networks

Martin Trapp

Turing.jl: A *robust*, *efficient* and *modular* library for general-purpose probabilistic AI.

`https://turing.ml`

Hong Ge[1], Kai Xu[2], Martin Trapp[3], Mohamed Tarek[4], Cameron Pfiffer[5], Tor Fjelde[1]

[1] University of Cambridge, [2] University of Edinburgh, [3]Graz Technical University, [4]UNSW Canberra, [5]University of Oregon

Probabilistic Machine Learning

Introduction

How can we make machines that learn from data using tools from probability theory?

Uncertainty is key to probabilistic machine learning and is expressed in all forms, e.g. noise in the data, uncertainty over the predictions, uncertainty over the model.

Probabilistic Inference

Challenge: Probabilistic inference is often difficult for interesting models.

	GANs	VAEs	Flows
Sampling	Y	Y	Y
Density	N	N/Y	Y
Marginals	N	N	?
Conditionals	N	N	?
Moments	N	N	?
MAP	N	N	?

Table 1: Robert Peharz, Sum-Product Networks and Deep Learning: A Love Marriage. ICML workshop on TPM, 2019.

Sum-Product Networks

Sum-Product Networks

- Sum-product networks (SPNs)[1] are a sub-class of so-called tractable probabilistic models or probabilistic circuits[2], that admit tractable probabilistic inference.
- A class of queries Q on a class of models M is tractable, iff for any query $q \in Q$ and model $m \in M$ the computational complexity is at most polynomial.
- SPNs admit many probabilistic inference tasks, such as marginalisation, in linear time in their representation size.

[1] H. Poon & P. Domingos: Sum-product networks: A new deep architecture. In UAI, 2011.

[2] Van den Broeck et al.: Tractable probabilistic models: Representations, algorithms, learning and applications. Tutorial at UAI, 2019.

Probabilistic Inference

	GANs	VAEs	Flows	SPNs
Sampling	Y	Y	Y	Y
Density	N	N/Y	Y	Y
Marginals	N	N	?	Y
Conditionals	N	N	?	Y
Moments	N	N	?	Y
MAP	N	N	?	N/Y

Table 2: Robert Peharz, Sum-Product Networks and Deep Learning: A Love Marriage. Talk at ICML, 2019.

What is a Sum-Product Network?

Let $\mathbf{X} = \{X_1, \ldots, X_D\}$ be set of random variables.

A probabilistic circuit over $\mathbf{X}$ is a tuple $\mathcal{S} = (\mathcal{G}, \psi, \theta)$, where

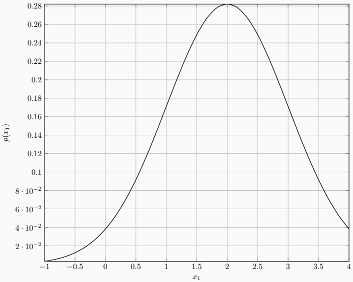
- $\mathcal{G}$ is a computational graph.
- ψ is a scope function.
- θ is a set of parameters, e.g. sum-weights and leaf node parameters.

A Sum-Product Network (SPN) is a *smooth* (complete) and *decomposable* probabilistic circuit.

Leaves L in $\mathcal{G}$

Leaf/input nodes are arbitrary (tractable) distributions,
e.g. Gaussian, Multinomial, variational autoencoder.

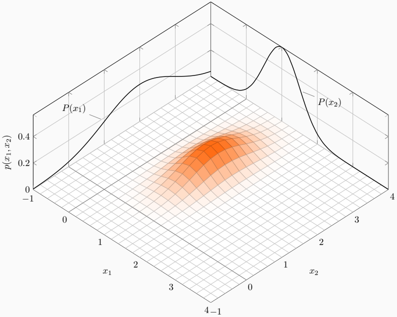
$$\mathsf{L}(x) = p(x \mid \theta_{\mathsf{L}})$$



Product Nodes P **in** $\mathcal{G}$

Product nodes encode independence assumptions between sets of random variables.

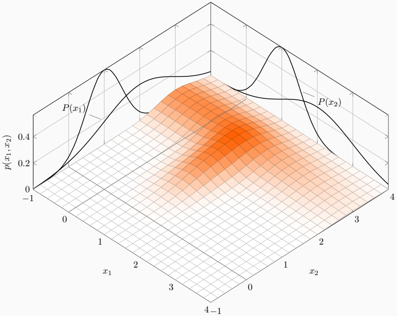
$$\mathsf{P}(x) = \prod_{\mathsf{C} \in \mathbf{ch}(\mathsf{P})} \mathsf{C}(x)$$



Sum Nodes S in $\mathcal{G}$

Sum nodes[3] replace independence with conditional independence within the network.

$$\mathsf{S}(x) = \sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{P})} w_{\mathsf{S},\mathsf{C}} \mathsf{C}(x)$$



[3] $\sum_{\mathsf{C} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{C}} = 1$ and $w_{\mathsf{S},\mathsf{C}} \geq 0$.

Computational Graph $\mathcal{G}$

$\mathcal{G}$ is a rooted connected directed acyclic graph (DAG), containing: sum (S), product (P) and leaf nodes (L).

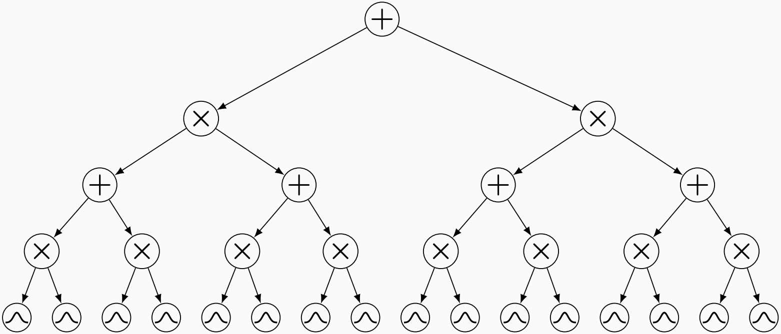


Figure 1: Example of a tree-shaped computational graph.

Scope Function ψ

The scope function assigning each node N in a sub-set of **X**,[4] and has to fulfil the following properties:

1. If N is the root node, then $\psi(\mathrm{N}) = \mathbf{X}$.
2. If N is a sum or product, then $\psi(\mathrm{N}) = \bigcup_{\mathrm{N}' \in \mathbf{ch}(\mathrm{N})} \psi(\mathrm{N}')$.

[4] This sub-set is often referred to as the scope of a node.

Scope Function ψ

The scope function assigning each node N in a sub-set of $\mathbf{X}$,[4] and has to fulfil the following properties:

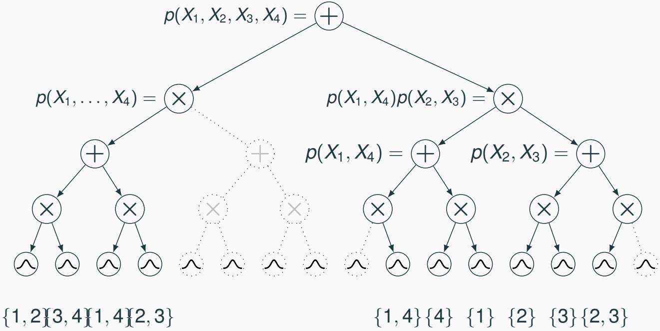
1. If N is the root node, then $\psi(\mathrm{N}) = \mathbf{X}$.
2. If N is a sum or product, then $\psi(\mathrm{N}) = \bigcup_{\mathrm{N}' \in \mathbf{ch}(\mathrm{N})} \psi(\mathrm{N}')$.

In case of SPNs we also assume that:

1. For each $\mathrm{S} \in \mathbf{S}$ we have $\forall \mathrm{N}, \mathrm{N}' \in \mathbf{ch}(\mathrm{S})\colon \psi(\mathrm{N}) = \psi(\mathrm{N}')$ (*smoothness*)
2. For each $\mathrm{P} \in \mathbf{P}$ we have $\forall \mathrm{N}, \mathrm{N}' \in \mathbf{ch}(\mathrm{P})\colon \psi(\mathrm{N}) \cap \psi(\mathrm{N}') = \emptyset$ (*decomposability*).

[4] This sub-set is often referred to as the scope of a node.

Example SPN $\mathcal{S} = (\mathcal{G}, \psi, \theta)$



Note that we define $\mathsf{L}(x) := 1$ for every x if and only if $\psi(\mathsf{L}) = \emptyset$.

Interpretations of SPNs

What is an SPN?

- Multi-linear feed-forward Neural Network (NN) with non-linear inputs.
- Deep structured mixture model with exponentially many components.
- Hierarchical tensor decomposition.

Learning Sum-Product Networks

Parameter Learning in SPNs

We can use backprop for parameter learning in SPNs.

More advanced approaches:

- Expectation Maximisation [R. Peharz et al.: On the latent variable interpretation of sum-product networks. TPAMI, 2017.]
- Variational Inference [H. Zhao et al.: Collapsed variational inference for sum-product networks. In ICML, 2016.]
- Bayesian moment matching [A. Rashwan et al.: Online and distributed Bayesian moment matching for parameter learning in SPNs. In AISTATS, 2016.]
- Safe Semi-Supervised Learning [M. Trapp et al.: Safe semi-supervised learning of sum-product networks. In UAI, 2017.]

Gradient-based optimisation in deep tree-structured SPNs has implicit acceleration effects. [M. Trapp et al.: Optimisation of overparametrized sum-product networks. ICML Workshop on TPM, 2019.]

Safe Semi-Supervised Learning - Example

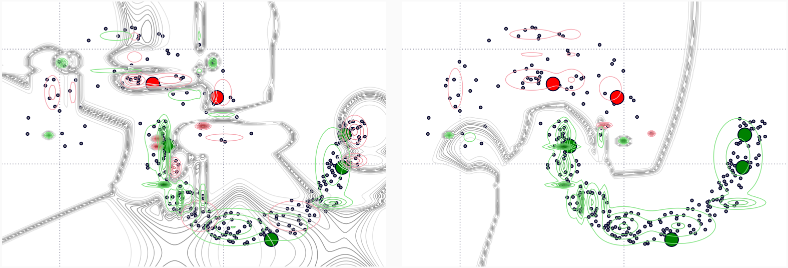


Figure 2: Decision boundary of a semi-supervised SPN at iteration 1 (left) and after convergence (right).

Challenges in Structure Learning

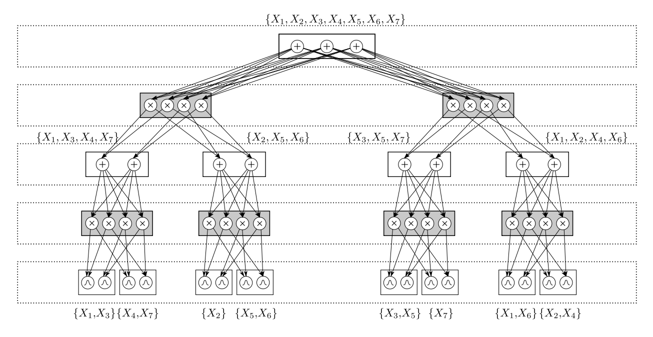
- The structure has to be smooth and decomposable, i.e., a sparsely connected graph.
- Structure learning has to be efficient.
- How to learn structures that generalise well, many approaches learn deep trees that are prune to overfitting.
- What is a good SPN structure? or What is a good principle to derive an SPN structure?

General-Purpose Learners (Selection)

- LearnSPN [R. Gens & P. Domingos: Learning the structure of sum-product networks. In ICML, 2013.]
- ID-SPN [A. Rooshenas & D. Lowd: Learning Sum-Product Networks with Direct and Indirect Variable Interactions. In ICML, 2014.]
- RAT-SPN [R. Peharz et al.: Random sum-product networks: A simple but effective approach to probabilistic deep learning. In UAI, 2019.]
- Bayesian SPN [M. Trapp et al.: Bayesian learning of sum-product networks. In NeurIPS, 2019.]

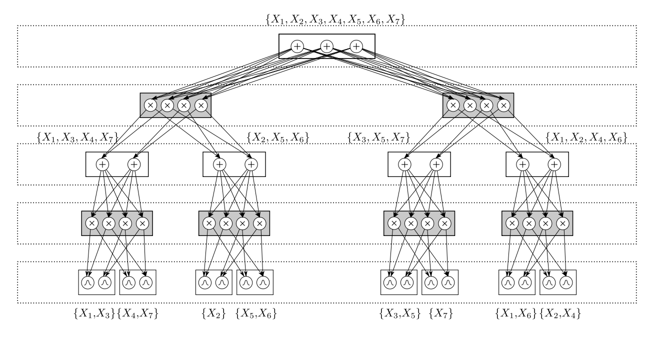
Random Structures

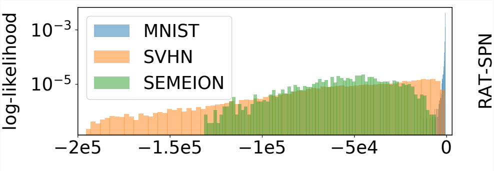
Idea: Use a large structure with random decompositions.



Random Structures

Idea: Use a large structure with random decompositions.





Why do we want Bayesian structure learning?

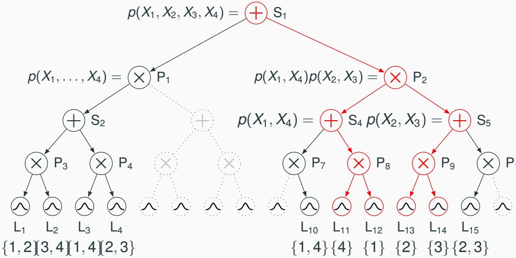
- Occam's razor effect prevents overfitting.
- Works on discrete, continuous and heterogeneous data domains.
- We can use nonparametric formulations, e.g. infinite SPNs, for continual learning.
- Structures can be inferred even in cases of missing values using exact marginalisation.

Bayesian Parameter Learning

The key insight for Bayesian parameter learning is that *sum nodes can be interpreted as latent variables*.

Bayesian Parameter Learning

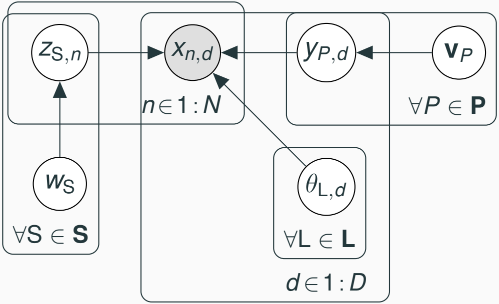
The key insight for Bayesian parameter learning is that *sum nodes can be interpreted as latent variables*.



$$\mathcal{T} \sim \prod_{(\mathsf{S},\mathsf{C}) \in \mathcal{T}_E} w_{\mathsf{S},\mathsf{C}}\,, \qquad \boldsymbol{x} \,|\, \mathcal{T} \sim \prod_{\mathsf{L} \in \mathcal{T}_V} p(x_{\psi(\mathsf{L})} \,|\, \theta_{\mathsf{L}}) \qquad (1)$$

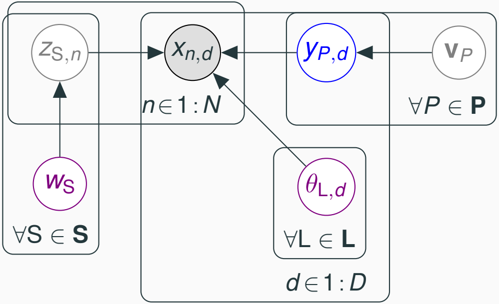
Bayesian Structure

Generative model for Bayesian learning of SPNs.



Bayesian Structure

Posterior inference using ancestral sampling within Gibbs.



Bayesian Structure - Missing Values Experiment

Performance under increasing amount of missing values.

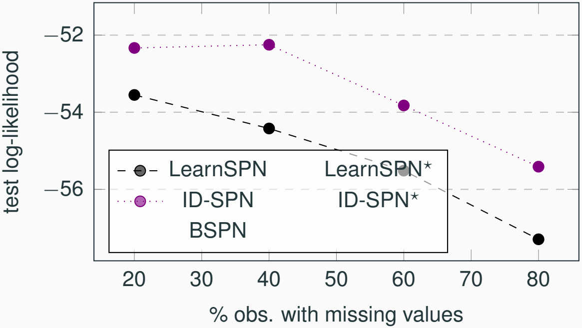


Figure 3: Results on EachMovie (D: 500, N: 5526) dataset.

Bayesian Structure - Missing Values Experiment

Performance under increasing amount of missing values.

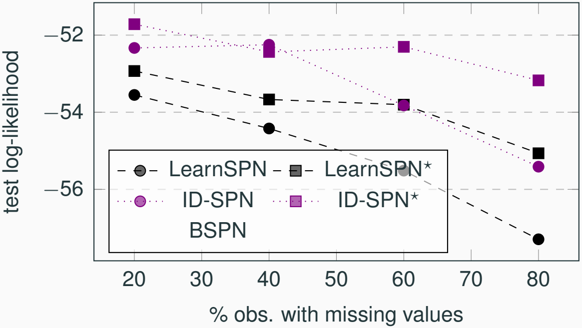


Figure 3: Results on EachMovie (D: 500, N: 5526) dataset.

Bayesian Structure - Missing Values Experiment

Performance under increasing amount of missing values.

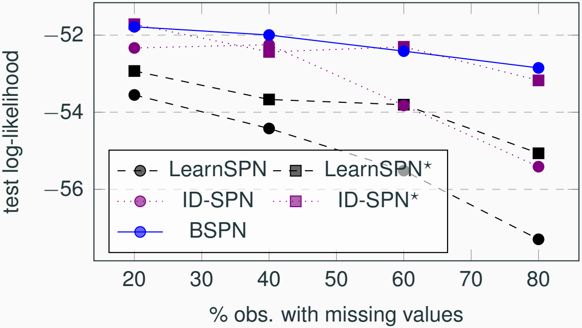


Figure 3: Results on EachMovie (D: 500, N: 5526) dataset.

Applications

Existing Applications

Some existing applications of SPNs:

- Computer vision, e.g. image classification, medical image processing, attend-infer-repeat.
- Language processing, e.g. language modelling, bandwidth extension.
- Robotics, e.g. semantic mapping.
- **Non-linear regression**, and many more[5]

[5] https://github.com/arranger1044/awesome-spn#applications

Gaussian Processes

A Gaussian Process (GP) is a collection of random variables indexed by an arbitrary covariate space $\mathcal{X}$, where any finite subset is Gaussian distributed.

A GP can be understood as a prior over functions.

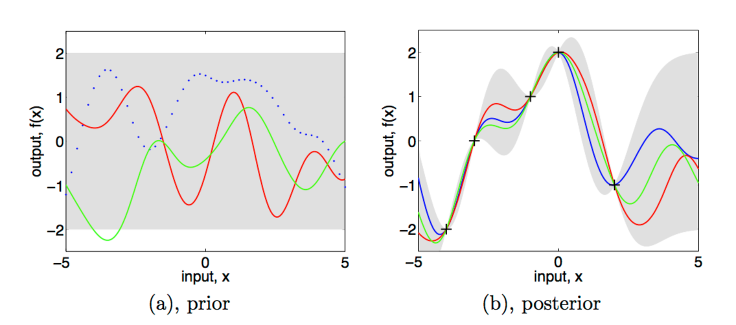


Figure 4: C. E. Rasmussen & C. K. I. Williams, Gaussian Processes for Machine Learning, 2006.

A GP is uniquely specified by a *mean-function* $m : \mathcal{X} \to \mathbb{R}$ and a *covariance function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.

The posterior predictive distribution (used for predictions) of a GP is Gaussian, i.e.,

$$p(f^* \mid \boldsymbol{f}) = \mathcal{N}\left(k_{\mathbf{x}^*,\mathbf{X}}^T k_{\mathbf{X},\mathbf{X}}^{-1} \boldsymbol{f}, k_{\mathbf{x}^*,\mathbf{x}^*} - k_{\mathbf{x}^*,\mathbf{X}} k_{\mathbf{X},\mathbf{X}}^{-1} k_{\mathbf{x}^*,\mathbf{X}}^T\right) \tag{2}$$

The inversion of $k_{\mathbf{X},\mathbf{X}}$ is usually computes using the Cholesky decomposition of $k_{\mathbf{X},\mathbf{X}}$?

A GP is uniquely specified by a *mean-function* $m : \mathcal{X} \to \mathbb{R}$ and a *covariance function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.

The posterior predictive distribution (used for predictions) of a GP is Gaussian, i.e.,

$$p(f^* \mid \boldsymbol{f}) = \mathcal{N}\left(k_{\mathbf{x}^*,\mathbf{X}}^T k_{\mathbf{X},\mathbf{X}}^{-1} \boldsymbol{f}, k_{\mathbf{x}^*,\mathbf{x}^*} - k_{\mathbf{x}^*,\mathbf{X}} k_{\mathbf{X},\mathbf{X}}^{-1} k_{\mathbf{x}^*,\mathbf{X}}^T\right) \tag{2}$$

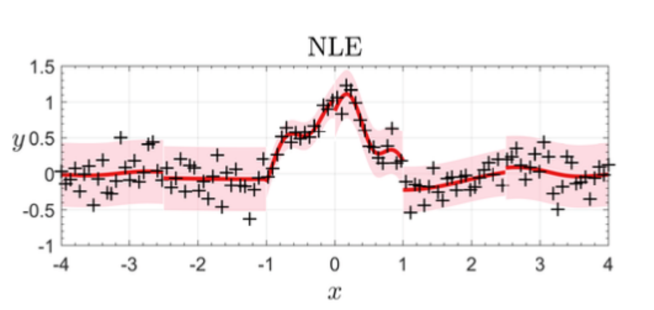
The inversion of $k_{\mathbf{X},\mathbf{X}}$ is usually computes using the Cholesky decomposition of $k_{\mathbf{X},\mathbf{X}}$?

Problem: The Cholesky decomposition scales $\mathcal{O}(N^3)$.

Local Experts

Local experts to approximate the GP or approximate the computation of predictions.

A natural way is to partition $\mathcal{X}$ into sub-sets $\mathcal{X}^{(k)}$, $k = 1, \ldots, K$. This is called the naive-local-experts model.



Local Experts

Existing solutions to discontinuities.

1. Product-of-Experts (PoE) / Bayesian Committee Machine (BCM)
   - Instead of partition $\mathcal{X}$, partition $\mathcal{X}$ into sub-sets $\mathcal{X}^{(k)}$.
   - Use an algorithm that works only on the sub-sets.
   - **Problem:** Not a stochastic process, results in over-conservative or over-confident estimates.
2. Mixture-of-Experts (MoE)
   - Use a gating network to assign observations to experts instead of hard boundaries.
   - Often intractable (due to the gating network).
3. Impose Continuity Constraints
   - Suffers from inconsistent variances and does not scale.

Why not use a large finite mixture of NLEs?

[6] M. Trapp et al.: Deep structure mixtures of Gaussian Processes. To appear in AISTATS, 2020.

Deep Structured Mixtures of Gaussian Processes

Why not use a large finite mixture of NLEs?

Deep Structured Mixture of GPs (DSMGP)[6]:

1. A DSMGP is an SPN (large structured mixture) over Gaussian measures.
2. DSMGPs perform exact Bayesian model averaging over a large set of NLEs.

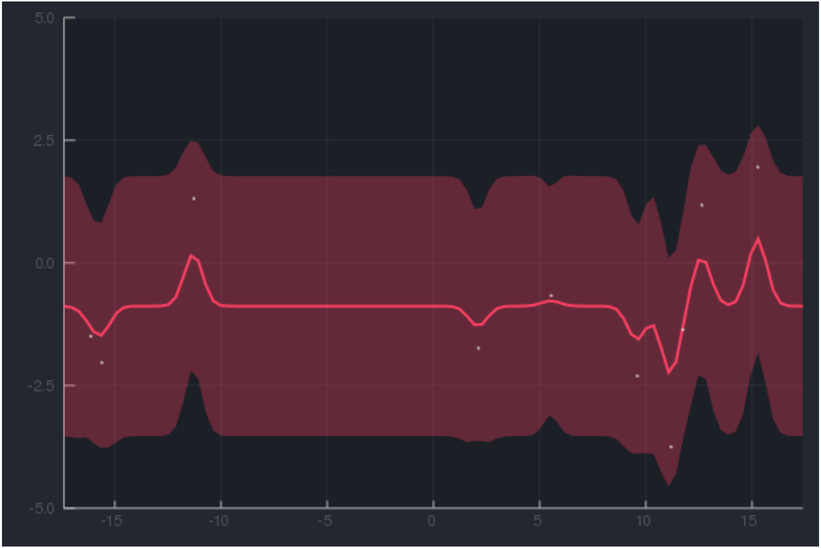
[6] M. Trapp et al.: Deep structure mixtures of Gaussian Processes. To appear in AISTATS, 2020.

Deep Structured Mixtures of Gaussian Processes

Benefits of DSMGPs:

- DSMGPs are a sound stochastic process.
- We can perform exact posterior inference in DSMGPs.
- DSMGPs have similar computational costs compared to other expert-based approaches.
- We can exploit the structure to perform Cholesky updates incrementally and share computations.
- DSMGPs capture predictive uncertainties consistently better.
- They do not suffer from severe discontinuities and can be used to model non-stationary data.

DSMGPs - Example



Thank you for your attention!