# Probabilistic Circuits: Tractable Representations for Learning and Reasoning

## Lecture 1 – Mixing Probabilistic Circuits with other Models

**Robert Peharz,**[1] **Antonio Vergari**[2]
[1]Graz University of Technology
[2]University of Edinburgh

**European Summer School on Artificial Intelligence**
Athens, 17th July 2024

# PCs and Probabilistic ML

## Probabilistic Circuits

+ tractable inference

- limited flexibility
- structurally constrained

## Common Probabilistic ML

+ flexible/expressive

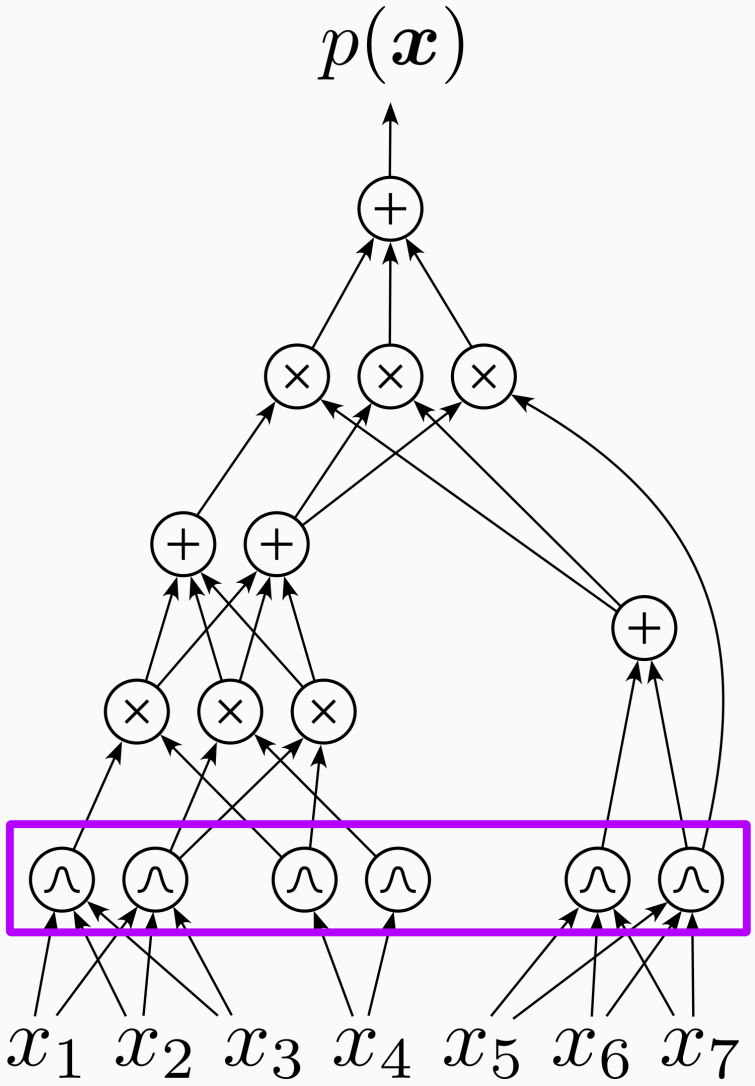
- intractable inference
- limited guarantees

**Let's break the brick wall and get best of both worlds!**

`https://github.com/smatmo/ESSAI24-PCs`

# Distribution Nodes

- input distributions should be **tractable**, i.e. facilitate marginalization, conditioning, expectations, maximization, etc.
- however, the PC is still well-defined and a proper model, **for any kind of leaf, even intractable ones!**
- we can use **entire probabilistic models** as leaves!



# PCs with Gaussian Processes

# PCs with Gaussian Processes

- **Gaussian processes** (GPs) are a prominent type of **distributions over functions**
- they are used in **Bayesian regression**, **Bayesian optimization**, **Bayesian quadrature**, etc.
- they are in fact tractable, as posterior inference scales $\mathcal{O}(N^3)$ where $N$ is the number of data points
- however, the **cubic cost can become prohibitive**, and approximations are needed
- here, we show how to come up with a hybrid between PCs and GPs, with better scalability properties!

Trapp et al., "Deep Structured Mixtures of Gaussian Processes"
`http://proceedings.mlr.press/v108/trapp20a.html`

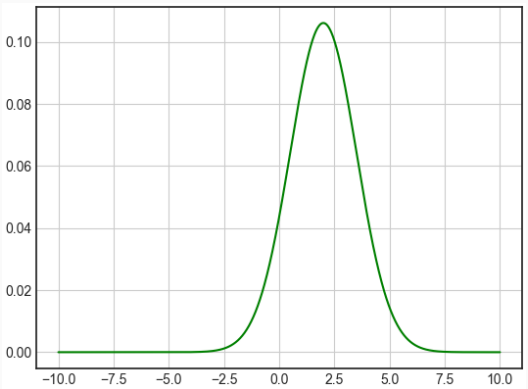
# Brief Intro to Gaussian Processes

# Gaussian Distribution

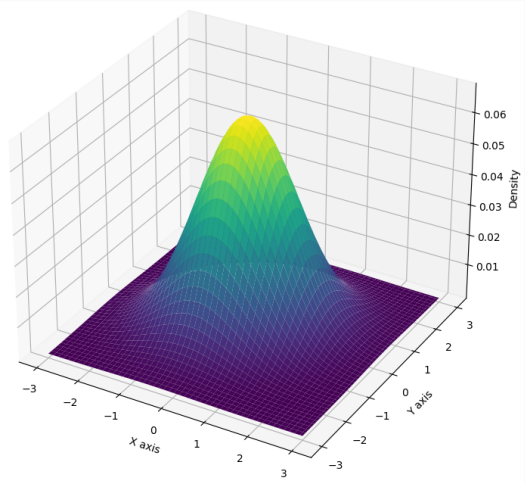
Gaussian density with $\boldsymbol{\mu}$ (mean vector) and $\boldsymbol{\Sigma}$ (covariance matrix)

$$p(\boldsymbol{X} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D \, |\boldsymbol{\Sigma}|}} \, e^{-\frac{1}{2}(\boldsymbol{X}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{X}-\boldsymbol{\mu})}$$

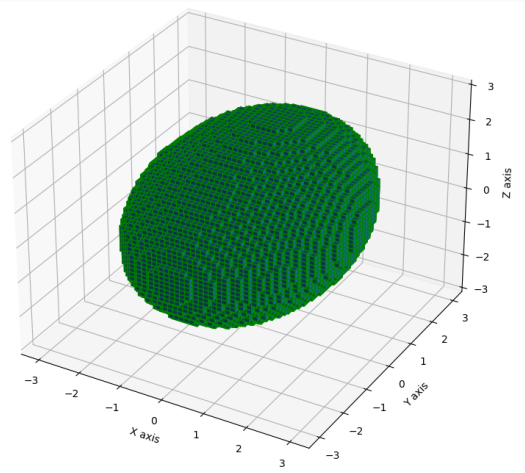
1D Gaussian




2D Gaussian




3D Gaussian



# Gaussian Distribution as a Random Function

Let $\boldsymbol{Y} = \{Y_1, Y_2, \ldots, Y_D\}$ be $D$ RVs, which are jointly Gaussian distributed. One way to interpret $\boldsymbol{Y}$ is as a **random function** $f$, mapping the **input space** $\{1, \ldots, D\}$ to real values.

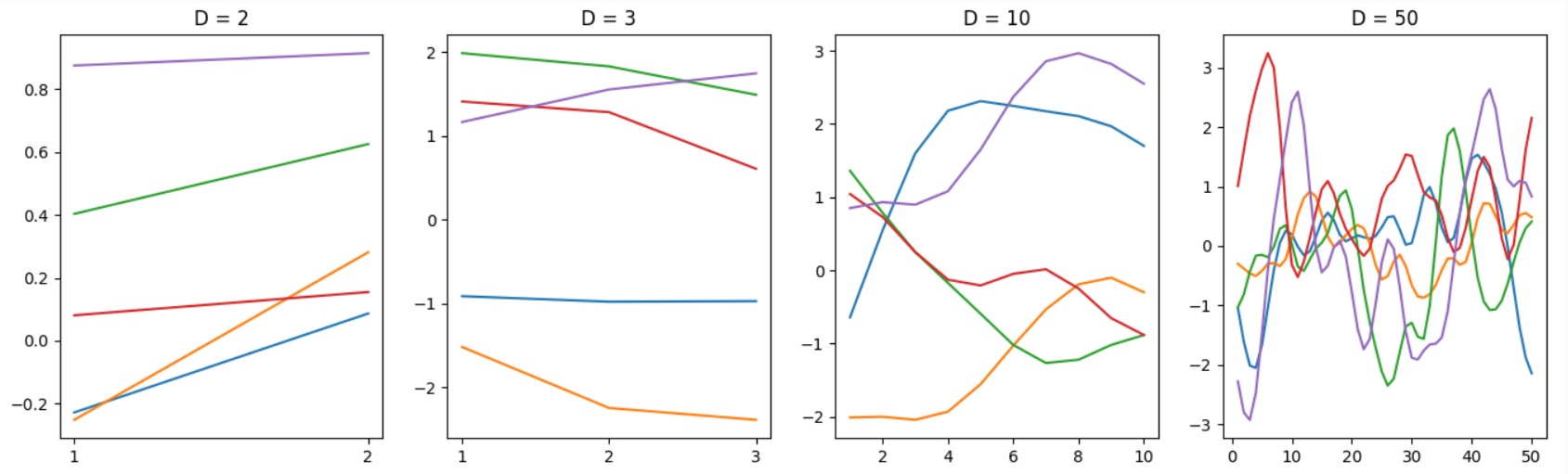
$$f : \{1, \ldots, D\} \mapsto \mathbb{R}$$

**input:** number $d$ between $1 \ldots D$

**output:** random real number $Y_d$

More precisely, note that the random variables $Y_1, \ldots, Y_D$ are functions defined on the same probability space, hence they map the atomic event $\omega$ to a vector $\boldsymbol{Y}(\omega) = (Y_1(\omega), \ldots, Y_D(\omega))^T$. The function $f$ should be though to depend on the random $\omega$, i.e. we might write $f_\omega$ to remind us of this.

Example draws of Gaussians with $D \in \{2, 3, 10, 50\}$ dimensions, interpreted as random functions from index set $\{1, \ldots, D\}$ to $\mathbb{R}$




The mean is 0 and we will learn how the covariance is defined. For now, note that the covariance is surely not diagonal (Why?).

# From Discrete to Continuous Inputs

- Gaussians with large $D$ indeed start to look like continuous random functions
- but, they are really only defined on the discrete **input space** $\mathcal{I} = \{1 \dots D\}$
- can we make the input space "really large", even infinite?
- can we take even an continuous index set like $\mathcal{I} = \mathbb{R}$?

## Gaussian Process

Let $\mathcal{X}$ be an **arbitrary** input space, e.g. $\mathcal{X} = \mathbb{R}, \mathbb{R}^3, \mathbb{R}^7, \mathbb{C}$. Let $\boldsymbol{Y} = \{Y_i\}_{i \in \mathcal{X}}$ be a collection of RVs indexed by $\mathcal{X}$ with the property that any **finite** subset of $\boldsymbol{Y}$ is multivariate Gaussian.

$\boldsymbol{Y}$ is called a **Gaussian process (GP)**.

The existence of GPs is highly non-trivial. After all, we need to make sure that an infinite collection of RVs defined on a common probability space exists, where such property holds.

However, rest assured that this is all good and well-defined.

Fun fact: a standard multivariate Gaussian is also a GP, defined on $\mathcal{X} = \{1, \ldots, D\}$.

A GP is uniquely specified via a

**mean function** $\mu : \mathcal{X} \mapsto \mathbb{R}$

**positive definite covariance function (kernel)*** $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$

Let $\mathcal{X}_n = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ by any finite subset of $\mathcal{X}$ with $n < \infty$ elements. Then

$$\boldsymbol{\mu} = (\mu(\boldsymbol{x}_1), \mu(\boldsymbol{x}_2), \ldots, \mu(\boldsymbol{x}_n)))^T$$

$$\Sigma = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \ldots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \ldots & k(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ \vdots & \ddots & \ddots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & k(\boldsymbol{x}_n, \boldsymbol{x}_2) & \ldots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix}$$

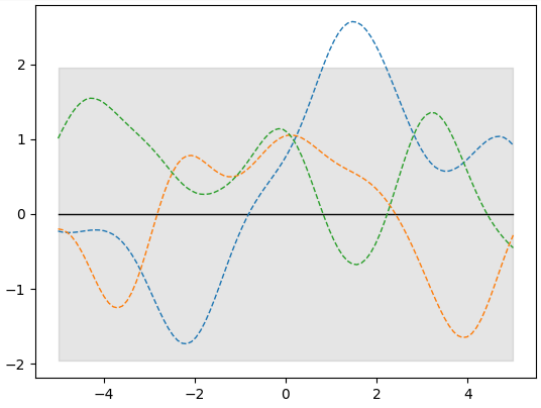
are the parameters of the Gaussian over $\boldsymbol{Y} = \{Y_i\}_{i \in \mathcal{X}_n}$

* meaning that $\Sigma$ is positive definite for any $\mathcal{X}_n$

# GPs with Various Kernels

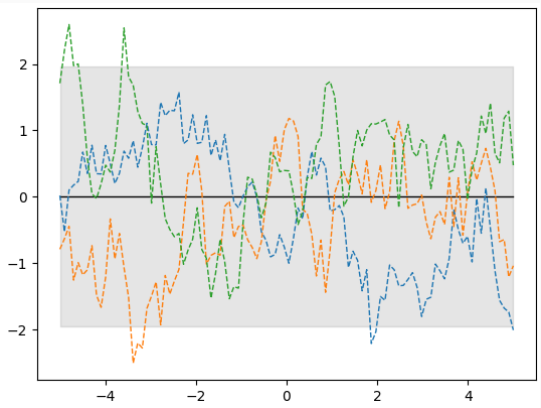
**Squared Exponential**

$$\sigma^2 \exp\left(\frac{-\frac{1}{2}\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{l^2}\right)$$



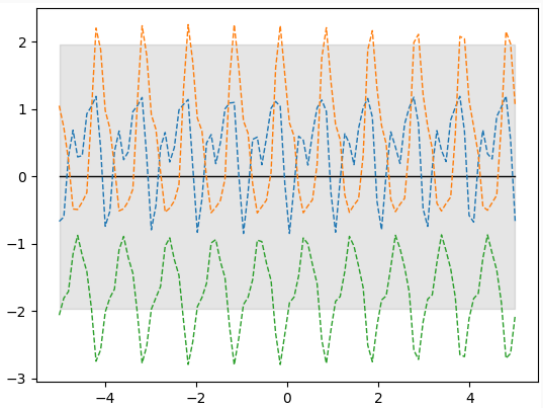
**Matérn $\nu = 0.5$**

$$\sigma^2 \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{l}\right)$$



**Sinusoidal**

$$\sigma^2 \exp\left(-2\frac{\sin(\pi\,\|\boldsymbol{x}-\boldsymbol{x}'\|/p)^2}{l^2}\right)$$



black line: GP mean

shaded area: 95% uncertainty interval

colored: different draws from GP

$\mathcal{X}$: input space (e.g. $\mathbb{R}$, $\mathbb{R}^D$)

$\mathcal{Y}$: output space (e.g. $\mathbb{R}$)

$\mathcal{D} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$: training data

**Goal:** learn a function $f : \mathcal{X} \mapsto \mathcal{Y}$ predicting $\boldsymbol{Y}$ from $\boldsymbol{X}$

## Regression

learn a **single** function $f$ by minimizing training loss

$$\min_f \mathcal{L}(\mathcal{D}, f)$$

## Bayesian Regression

posterior inference over **all** $f$

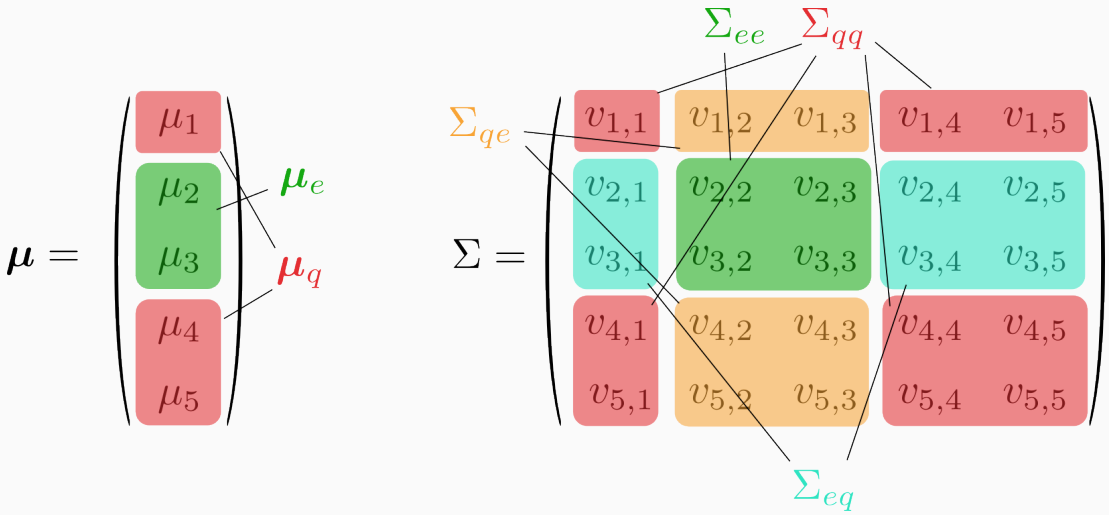
$$\textbf{prior} \quad p(f) \qquad \textbf{posterior} \quad p(f \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid f)\ p(f)}{p(\mathcal{D})}$$

# Bayesian Regression with GPs

**Bayesian inference over whole functions seems hard, but it is tractable with GPs!**

When the prior $p(f)$ is a GP, the posterior $p(f \mid \mathcal{D})$ is again a GP with closed form mean and covariance function. In practice, it reduces to **finite Gaussian conditionals**. Recall from Lecture 1:

$$\boldsymbol{\mu}_{q|e} = \boldsymbol{\mu}_q + \Sigma_{qe}\Sigma_{ee}^{-1}(\boldsymbol{x}_e - \boldsymbol{\mu}_e)$$
$$\Sigma_{q|e} = \Sigma_{qq} - \Sigma_{qe}\Sigma_{ee}^{-1}\Sigma_{eq}$$

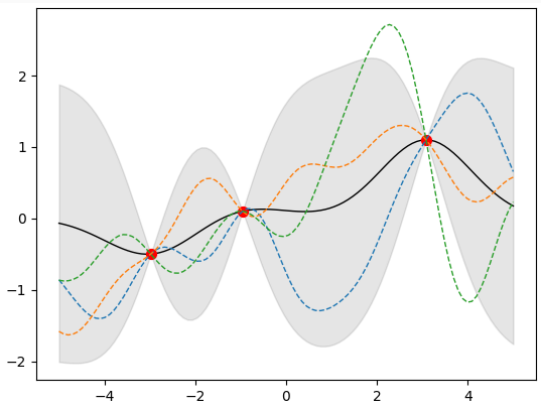
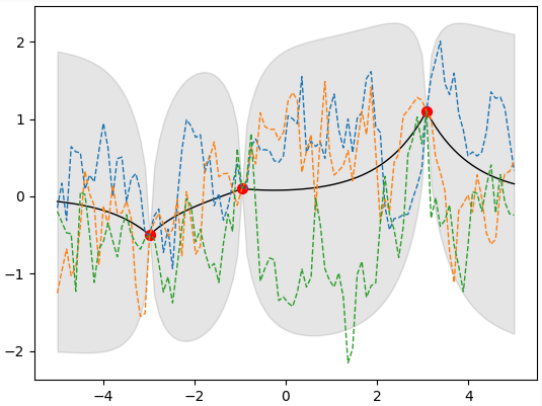
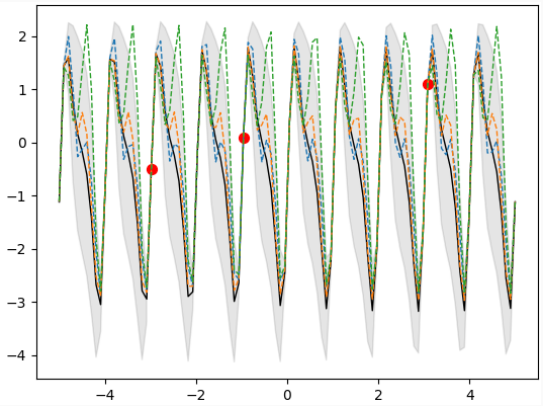


# Bayesian Regression with GPs cont'd

In the context of Bayesian regression,

- the **evidence** variables are the $y$ values in the training data
- the $\boldsymbol{x}$ values in the training data determine where the mean and covariance functions are evaluated
- the **query** variable is the $Y$ value at any test point $\boldsymbol{x}^*$

**Squared Exponential** **Matérn** $\nu = 0.5$ **Sinusoidal**

# Probabilistic Circuits and GPs

# Basic Idea

Posterior inference reduces to finite Gaussian conditionals:

$$\boldsymbol{\mu}_{q|e} = \boldsymbol{\mu}_q + \Sigma_{qe}\Sigma_{ee}^{-1}(\boldsymbol{x}_e - \boldsymbol{\mu}_e)$$
$$\Sigma_{q|e} = \Sigma_{qq} - \Sigma_{qe}\Sigma_{ee}^{-1}\Sigma_{eq}$$

**Bottleneck**

- $\Sigma_{ee}$ is an $N \times N$ matrix, containing the kernel evaluations between all pair of training inputs
- matrix inversion of $\Sigma_{ee}$ scales cubically in the number of rows (columns), i.e. $\mathcal{O}(N^3)$

**Idea**

- chop up the dataset into many small ones (with overlap)
- learn local GPs and glue them together via the PC language
- **basically, PCs with GP leaves!**

# Insight 1: Mixtures of GP Distributions

Let $\mathcal{X}$ be an input space (e.g. $\mathcal{X} = \mathbb{R}$) and $p_1(f), \ldots, p_K(f)$ be $K$ different GP distributions over functions defined on $\mathcal{X}$. Then a convex combination

$$p(f) = \sum_{k=1}^{K} w_k \, p_k(f), \quad w_k \geq 0, \; \sum_k w_k = 1$$

is also a distribution over functions defined on $\mathcal{X}$, denoted as **Mixture of GP Distributions**.

The notation is not entirely rigorous, since GPs do not have a density in the common sense. However, the above is true when evaluating arbitrarily many—but finitely many—input positions of $f$. Also, when interpreting $p$ and $p_k$ as probability measure of stochastic processes, the notation is correct.

# Mixing GPs vs. Mixing GP Distributions

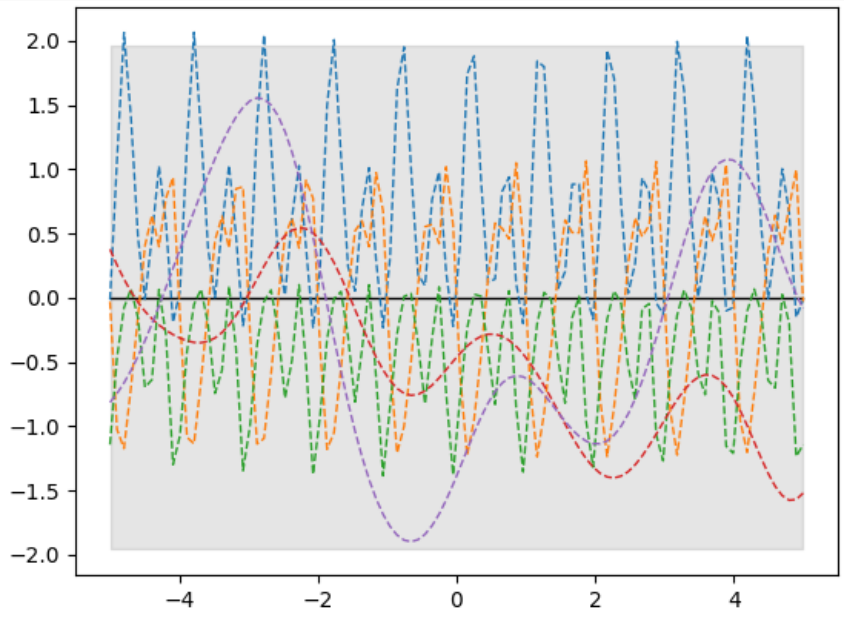
**Note** that a mixture of GP **distributions** is not the same as a mixture of GPs!

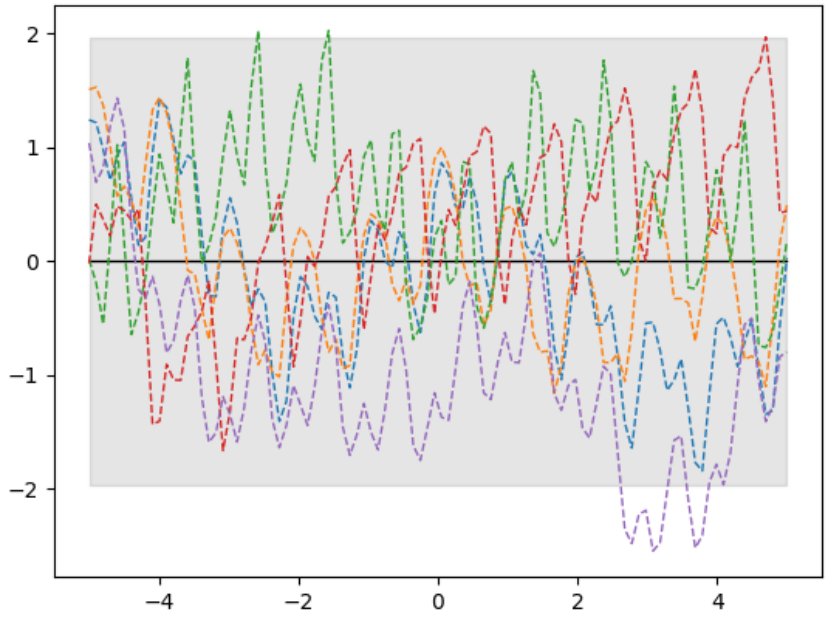
- The latter—**usually done in literature—mixes GPs, i.e. the random functions**. The result is still a GP (it turns out, that one simply needs to mix the kernels)! Draws will look like superpositions of draws from the original GPs.
- The former—**proposed by Trapp et al.—mixes the GP distributions**. A mixture of GP distributions is **not a GP**! Just like a mixture of Gaussians is not Gaussian any more. Independent draws from $p(f)$ will look like draws from one of the $p_k(f)$, occurring with probability $w_k$.
- However, **mixtures of mixtures of GP distributions** are again **mixtures of GP distributions**.

# Who's who?

Example

Here 5 independent draws of two different approaches are demonstrated with squared exponential and sinusoidal kernels, both mixed with weight 0.5. Who's who?

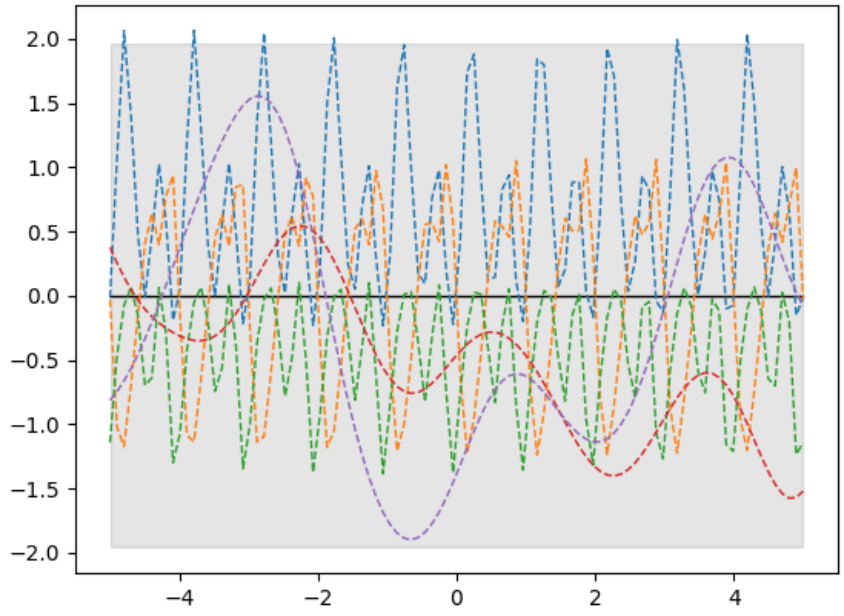


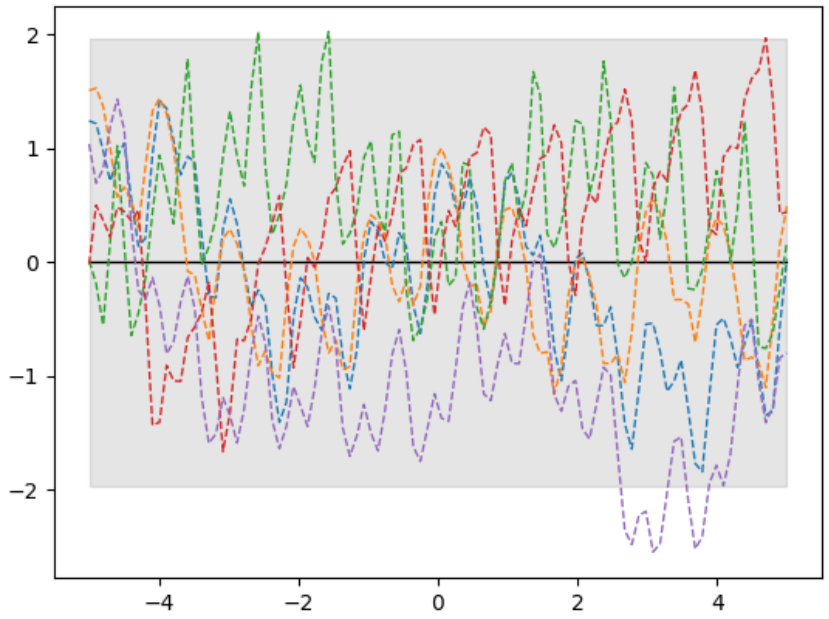
# Who's who?

Here 5 independent draws of two different approaches are demonstrated with squared exponential and sinusoidal kernels, both mixed with weight 0.5. Who's who?

## Mixture of GP Distributions




## Mixture of GPs



# Insight 2: Decomposable Products of GP Distributions

Let $\mathcal{X}_1$ and $\mathcal{X}_2$ be **disjoint** input spaces and let $p_1(f)$ and $p_2(g)$ be (mixtures of) GP distributions over functions $f\colon \mathcal{X}_1 \mapsto \mathbb{R}$ and $g\colon \mathcal{X}_2 \mapsto \mathbb{R}$, respectively. Then

$$p([f, g]) = p(f) \times p(g)$$

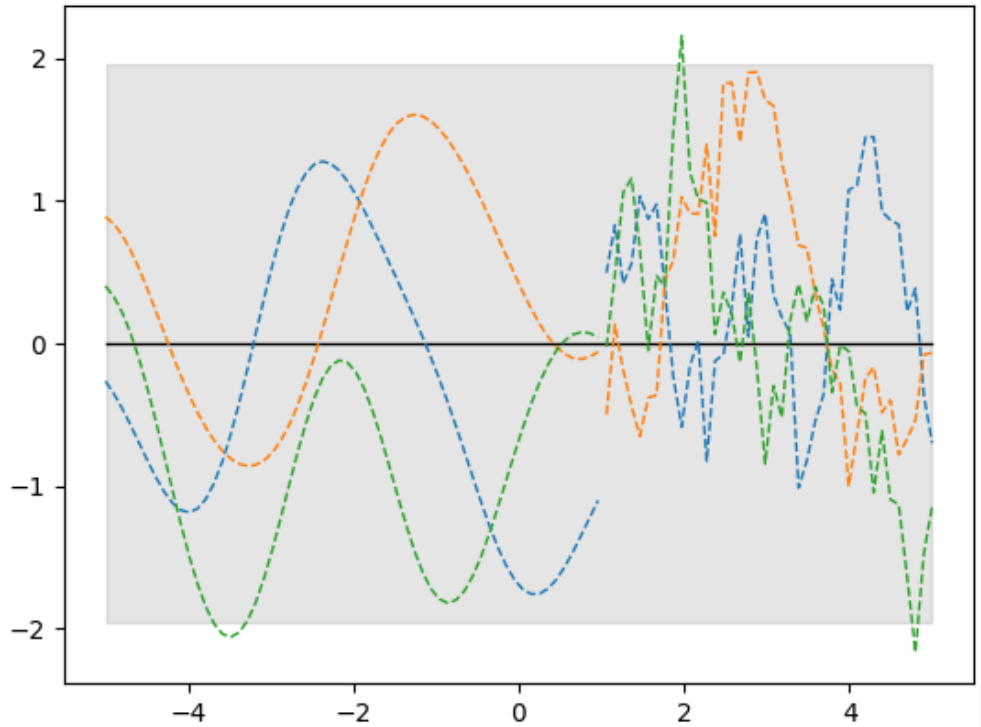
is a (mixture of) GP distributions over functions $[f, g]\colon \mathcal{X}_1 \cup \mathcal{X}_2 \mapsto \mathbb{R}$ .

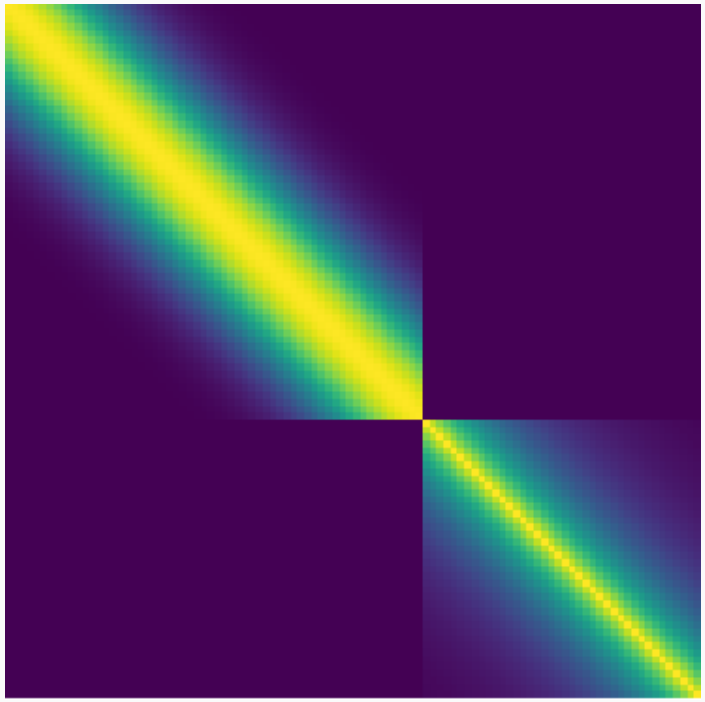
The concept carries over to arbitrarily many input spaces $\mathcal{X}_1, \ldots, \mathcal{X}_K$.

Again, we should argue in terms of probability measures. However, we might simply think in evaluation of densities at finitely many indices (inputs).

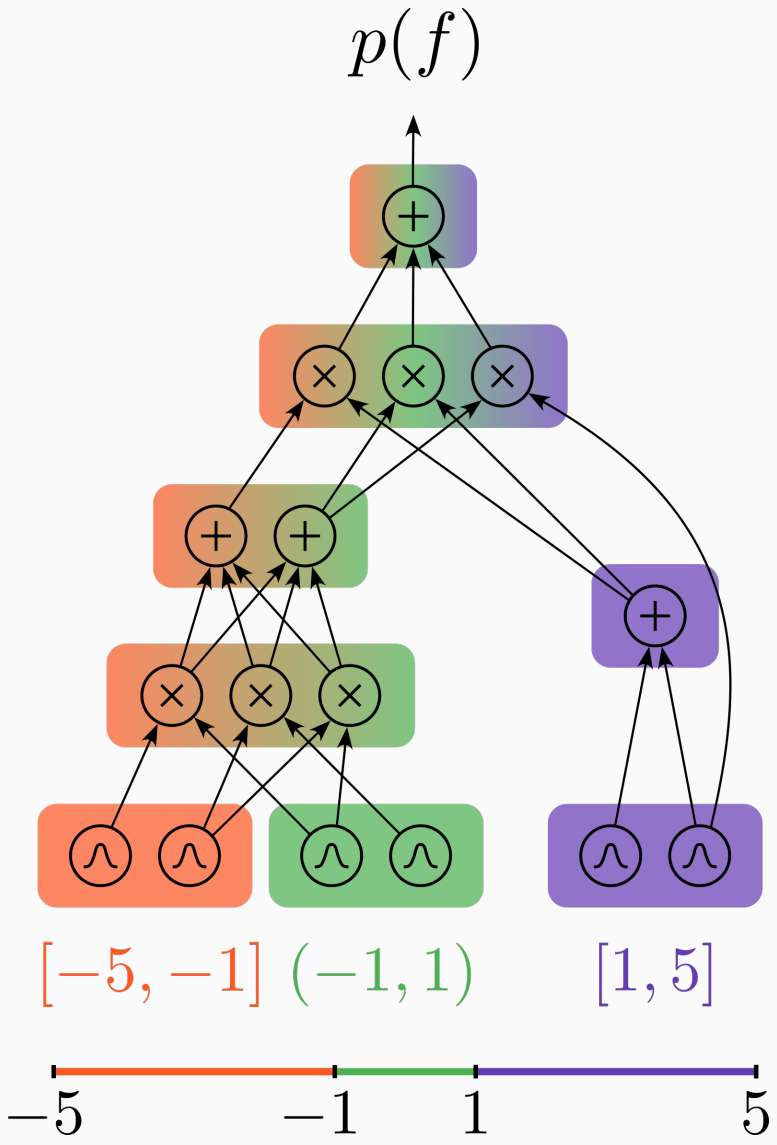
Product of two GPs with **squared exponential** and **Mátern kernel**, defined on $\mathcal{X}_1 = [-5, 1]$ and $\mathcal{X}_2 = (1, 5]$, respectively.

The draws look like stitched draws from the two original GPs. The resulting covariance function is block diagonal.



- basically, our job is done
- **input distributions are now GPs**
- hence, their scope contains now infinitely many RVs
- we still require **smoothness** and **decomposability**
- such a **GPPC** represents a "deep mixture of factorized GPs", a distribution $p(f)$ over functions
- right: example for a GPPC over the interval $[-5, 5]$



# Posterior Inference

- the power of GPs is that **posterior inference is tractable:**

$$\textbf{prior} \quad p(f) \qquad \textbf{posterior} \quad p(f \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid f)\ p(f)}{p(\mathcal{D})}$$

- specifically, the posterior is again a GP with mean and covariance

$$\begin{aligned} \boldsymbol{\mu}_{q|e} &= \boldsymbol{\mu}_q + \Sigma_{qe}\Sigma_{ee}^{-1}(\boldsymbol{x}_e - \boldsymbol{\mu}_e) \\ \Sigma_{q|e} &= \Sigma_{qq} - \Sigma_{qe}\Sigma_{ee}^{-1}\Sigma_{eq} \end{aligned}$$

- similarly, the posterior of a GPPC is again a GPPC!

## Conditional PC

Let a PC over $\boldsymbol{X}$ be given and let $\boldsymbol{X}_q \subseteq \boldsymbol{X}$ be query variables and $\boldsymbol{X}_e = \boldsymbol{X} \setminus \boldsymbol{X}_q$.

- for each sum with weights $w_k$ and input nodes $p_k$, replace the weights with

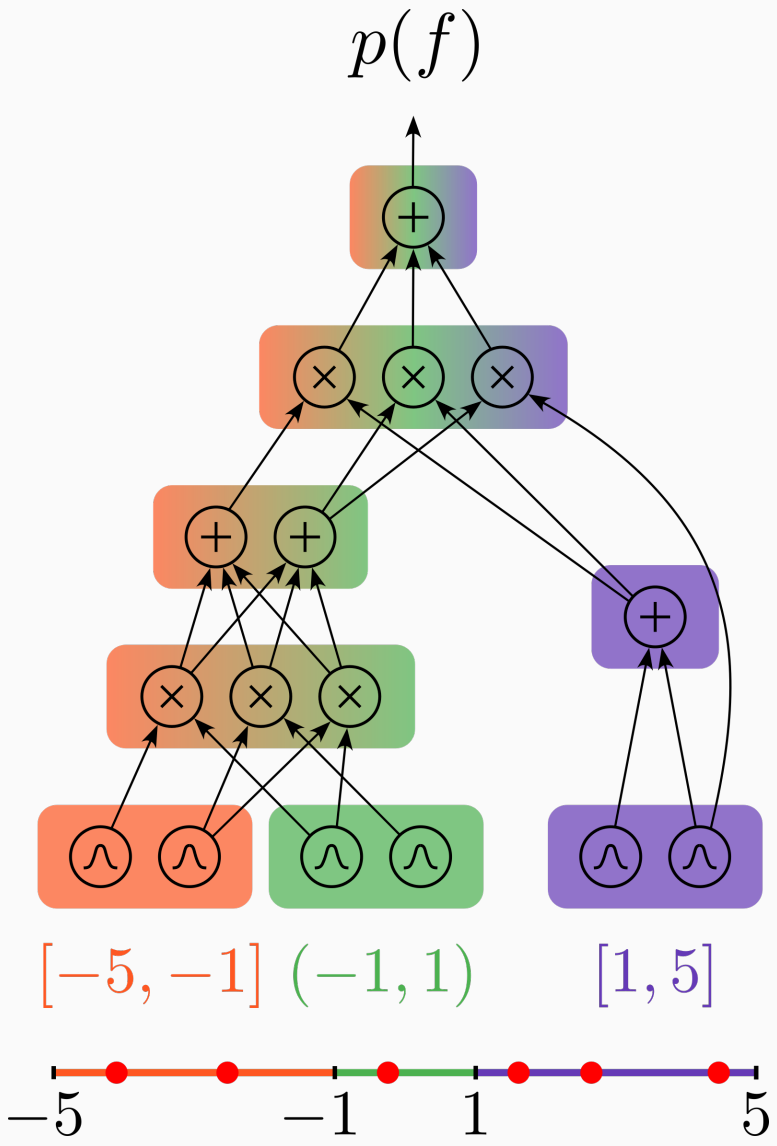
$$\tilde{w}_k = \frac{w_k \, p_k(\boldsymbol{x}_e)}{\sum_i w_i \, p_i(\boldsymbol{x}_e)}$$

  where $p_k(\boldsymbol{x}_e)$ has been computed by the marginal PC.
- replace each input distribution $D$ with the corresponding conditional, with query variables $\boldsymbol{X}_q \cap sc(D)$ and evidence variables $\boldsymbol{X}_e \cap sc(D)$.

**This works the same in GPPCs.**

# Posterior Inference in GPPCs

- say we observe a dataset $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
- every GP evaluates the probability of the points which fall in their scope
- example: the orange, green, and purple GPs evaluate the density of 2d, 1d and 3d Gaussians, respectively, on the Y-values corresponding to the points in their scope
- propagate upwards through the PC (as usual)

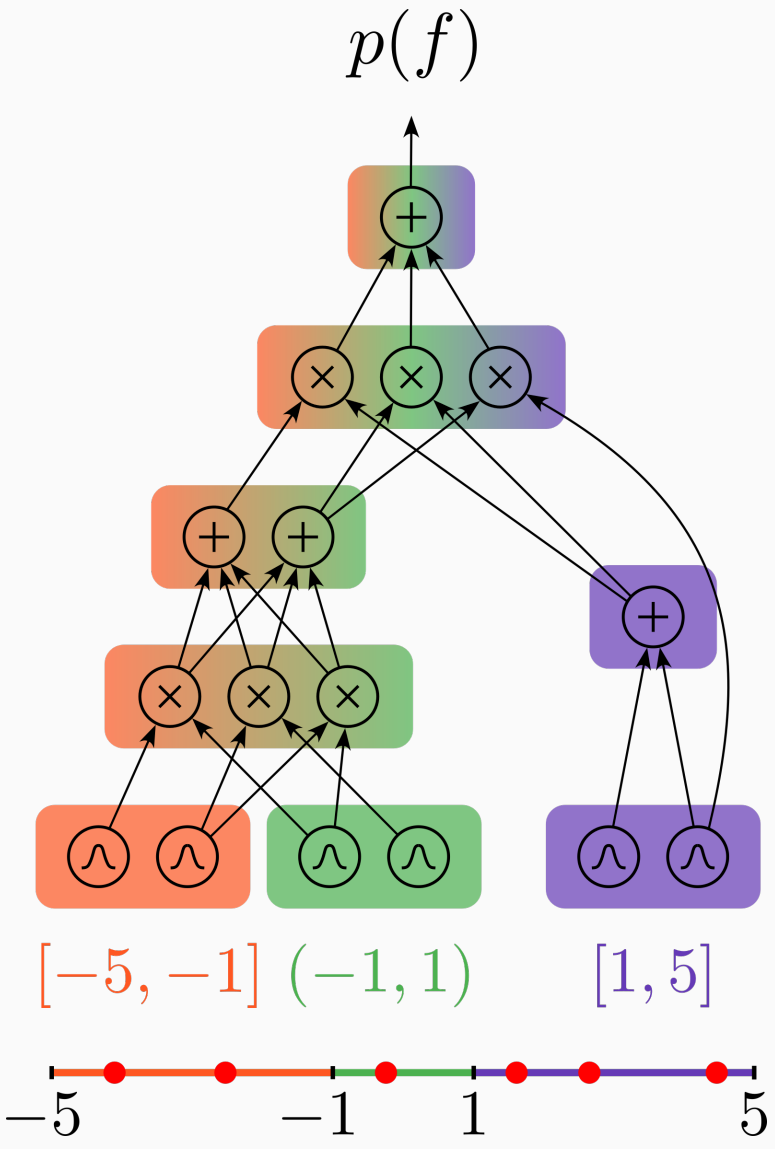


# Posterior Inference in GPPCs

- for each sum with weights $w_k$ and input nodes $p_k$, replace the weights with

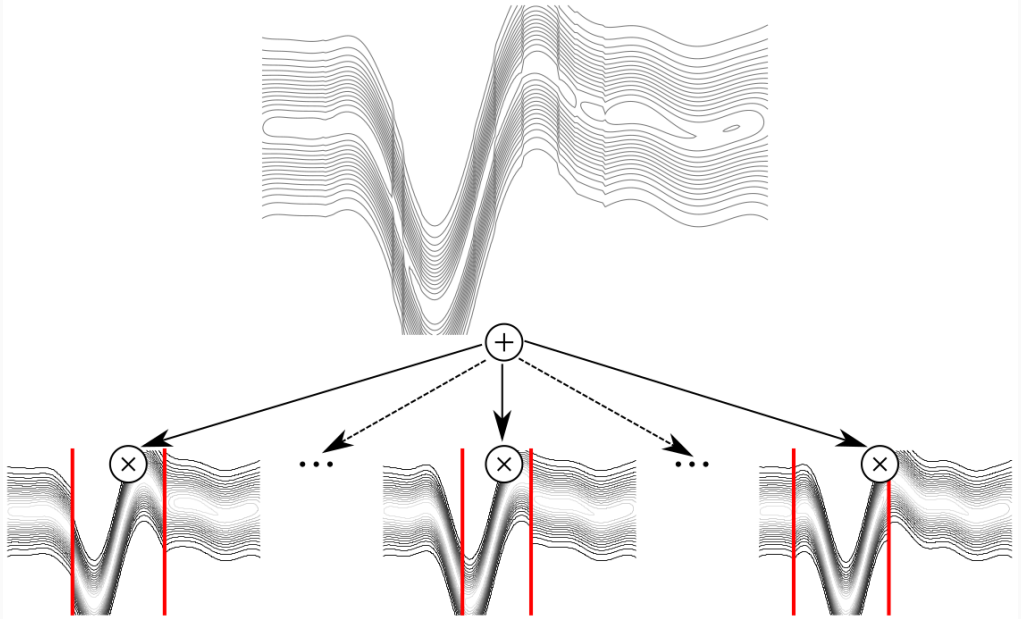
$$\tilde{w}_k = \frac{w_k \, p_k(\boldsymbol{x}_e)}{\sum_i w_i \, p_i(\boldsymbol{x}_e)}$$

- replace each GP in the input layer with the posterior GP, based only on the points in their scope
- result is the **exact posterior GPPC**



# Structure of GPPCs

In the example before, the entire input space $\mathcal{X}$ was split into disjoint parts in a unique way. This is not necessary. Scopes can overlap as long as smoothness and decomposability are maintained. For example, for a GPPC of depth 1:

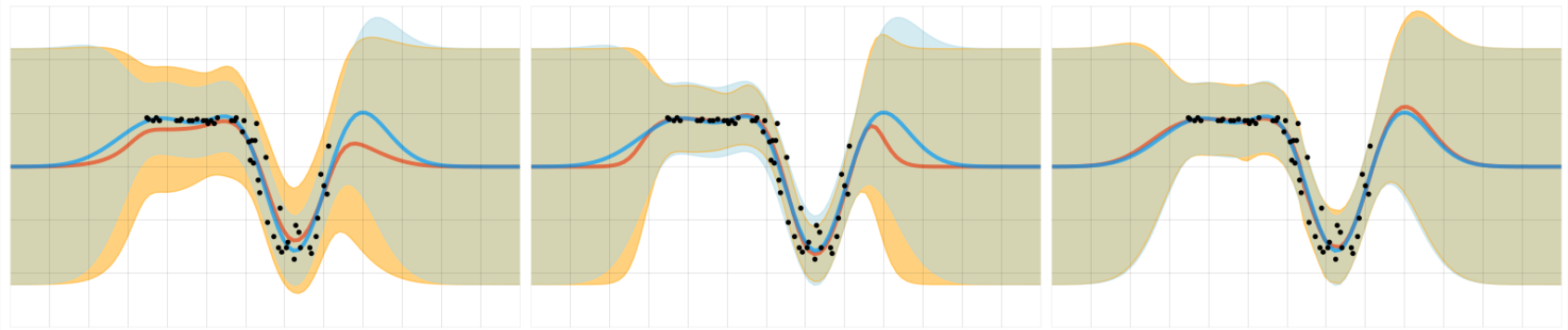


The cutting points might be selected randomly. We can cut recursively, yielding a randomized deep structure for GPPCs.

# GPPCs as Tractable Divide-and-Conquer Approach

- the main benefit of GPPCs is that by chopping up the input space, every GP leaf gets fewer data points
- we have now **many GP leaves** instead of just **one GP**
- this overhead gets easily amortized due to the cubic cost of inference
- moreover, GP leaves which share data points can **share computation** for the matrix inverse (shared **Cholesky decomposition**)
- there are other GP approaches with the idea of divide-and-conquer, which, however:
  - are not a sound probabilistic model,
  - sound but intractable
- GPPCs on the other are **sound distributions over functions** and **admit tractable inference**

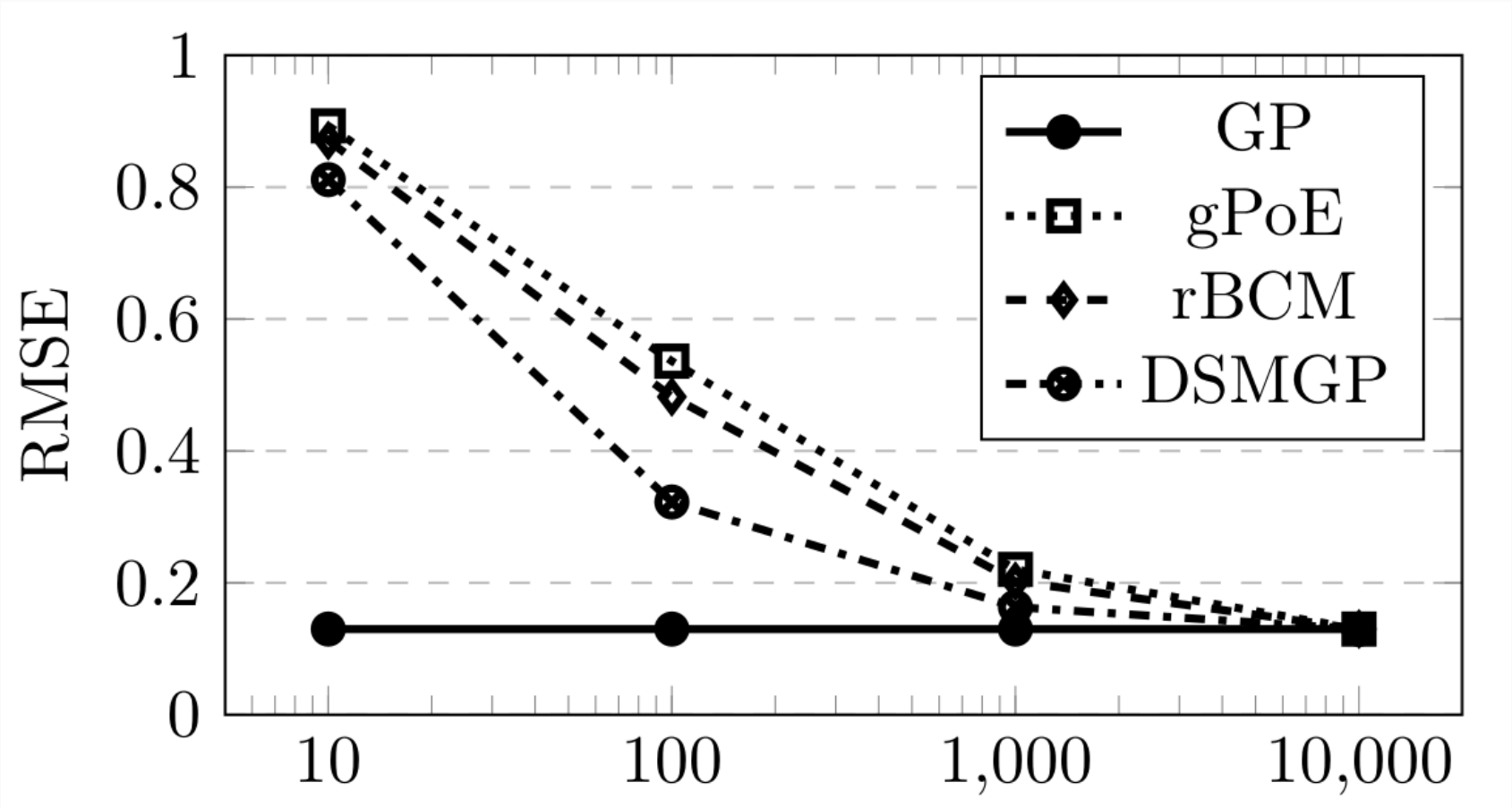
# Fit and Uncertainties



(a) generalized PoE (b) robust BCM (c) DSMGP (our work)

Qualitative comparison between **generalized product of experts (gPoE)**, **Bayesian committee machine** (BCM) and a GPPC (called DSMGP here). All methods divide the data in similar portions and use the same kernel (SE). Overlaid are mean and variance of an exact GP with the same kernel.

# Approximation Error




Approximation error on the `Kin40k` dataset, as a function of samples allocated to GP experts. GPPC is called DSMGP called here.

# Benchmark Performance

| Dataset | | Const. | LR | GP | SVGP | KISS | gPoE | rBCM | **DSMGP** |
|---|---|---|---|---|---|---|---|---|---|
| **Airfoil** | MAE | 0.82 | 0.53 | 0.50 | **0.32** | 0.51 | 0.35 | 0.34 | **0.32** |
| | NLPD | 1.43 | 1.05 | 0.99 | 0.59 | 1.00 | 0.72 | 3.21 | **0.57** |
| **Parkin.** | MAE | 0.85 | 0.82 | 0.78 | **0.68** | 0.78 | 0.84 | 0.80 | 0.74 |
| | NLPD | 2.88 | 2.79 | 2.73 | **2.52** | 2.73 | 4.49 | 3.80 | 2.66 |
| **Kin40K** | MAE | 0.81 | 0.81 | 0.79 | **0.25** | 0.79 | 0.80 | 0.43 | 0.78 |
| | NLPD | 1.42 | 1.42 | 1.39 | **0.37** | 1.39 | 2.68 | 4.14 | 1.38 |
| **House** | MAE | 0.62 | 0.49 | NA | **0.39** | 0.43 | 0.50 | 0.40 | **0.39** |
| | NLPD | 1.45 | 1.30 | NA | **1.06** | 1.10 | 4.61 | 4.58 | 1.11 |
| **Protein** | MAE | 0.89 | 0.71 | NA | 0.57 | 0.64 | 0.82 | 0.70 | **0.55** |
| | NLPD | 1.41 | 1.25 | NA | **1.11** | 1.19 | 2.38 | 4.57 | **1.11** |
| **Year** | MAE | 0.74 | 0.73 | NA | **0.57** | NA | 0.74 | 0.74 | 0.72 |
| | NLPD | 1.41 | 1.39 | NA | **1.21** | NA | 3.78 | 1.49 | 1.38 |
| **Flight** | MAE | 0.56 | **0.54** | NA | **0.54** | NA | 0.56 | 0.56 | **0.54** |
| | NLPD | 2.87 | 2.85 | NA | **2.80** | NA | 8.05 | 11.51 | 2.84 |

Mean absolute error and negative log-predictive density. GPPC is called DSMGP called here.

# PCs and Continuous Mixtures

# PCs and Variational Autoencoders

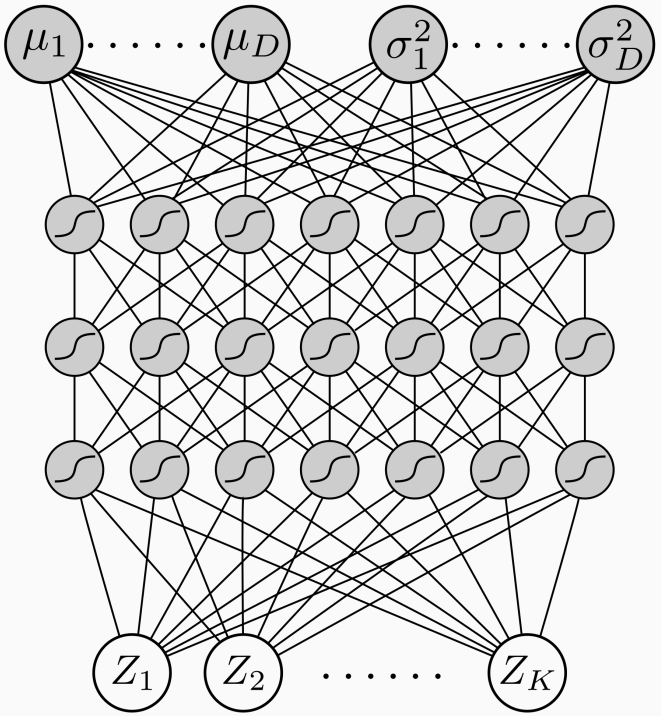
- variational autoencoders (VAEs) are a prominent deep generative model
- they are relatives of PCs:
  - PCs are deep structured **finite** mixture models
  - VAEs are **infinite** (continuous) mixture models
- crucially, PCs allow **tractable inference**, while **inference in VAEs is intractable**
- still, it makes sense to mix the two to get best of both worlds

Tan and Peharz, Hierarchical decompositional mixtures of variational autoencoders, ICML 2019
Stelzner et al., Faster attend-infer-repeat with tractable probabilistic models, ICML 2019
Correia et al., Continuous Mixtures of Tractable Probabilistic Models, AAAI'23
Gala et al., Probabilistic Integral Circuits, AISTATS'24
Gala et al., Scaling Continuous Latent Variable Models as Probabilistic Integral

- let $\boldsymbol{Z} = (Z_1, \ldots, Z_K)$ be a **continuous latent** RV, typically white Gaussian
- $\boldsymbol{Z}$ gets transformed by a neural network (**decoder**), returning
  - $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_D)$
  - $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_D)$
- $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ parametrize a diagonal Gaussian over data $\boldsymbol{X}$:

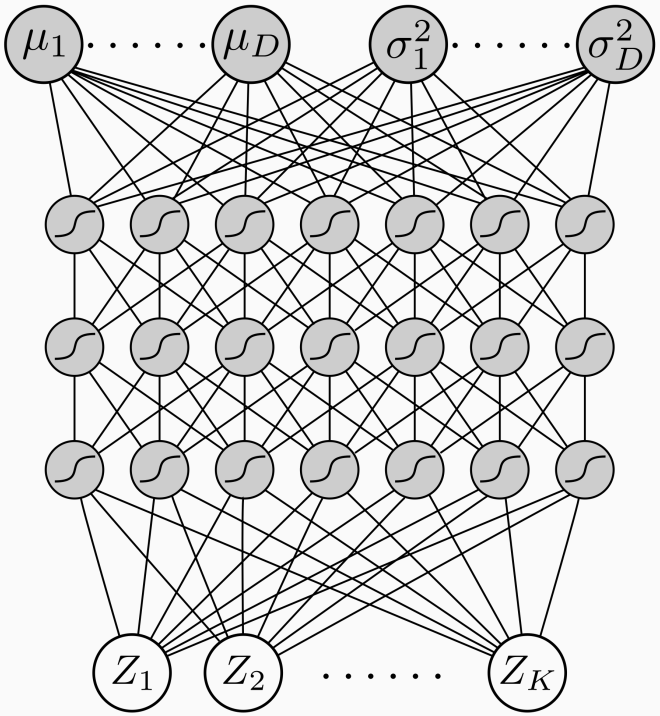
$$p(\boldsymbol{X} \mid \boldsymbol{Z}) = p_{\text{gauss}}(\boldsymbol{X} \mid \boldsymbol{\mu}, \mathit{diag}(\boldsymbol{\sigma}))$$

- $\boldsymbol{X}$ is conditional on $\boldsymbol{Z}$, because $\boldsymbol{\mu}(\boldsymbol{Z})$ and $\boldsymbol{\sigma}(\boldsymbol{Z})$ are functions of $\boldsymbol{Z}$



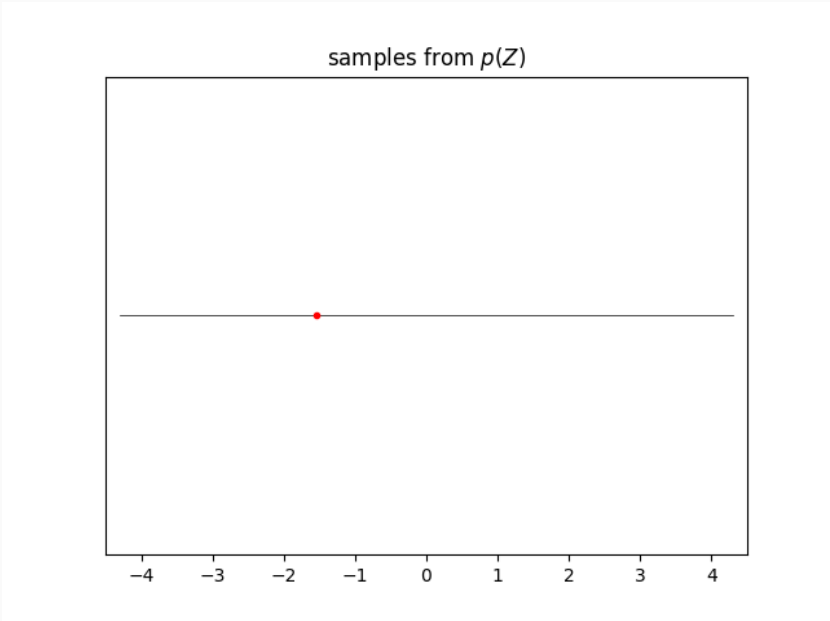
- sampling data works as follows:
  - $z \sim p_{\text{gauss}}(\boldsymbol{Z} \mid \mathbf{0}, \boldsymbol{I})$
  - input $\boldsymbol{z}$ to decoder $\rightarrow \boldsymbol{\mu}$ and $\boldsymbol{\sigma}$
  - $\boldsymbol{x} \sim p_{\text{gauss}}(\boldsymbol{X} \mid \boldsymbol{\mu}, \mathit{diag}(\boldsymbol{\sigma}))$
  - return $\boldsymbol{x}$, throw away $\boldsymbol{z}$
- hence, every **observed** data sample $\boldsymbol{x}_i$ is assumed to be generated from a **latent code** $\boldsymbol{z}_i$
- model is a **continuous mixture**:

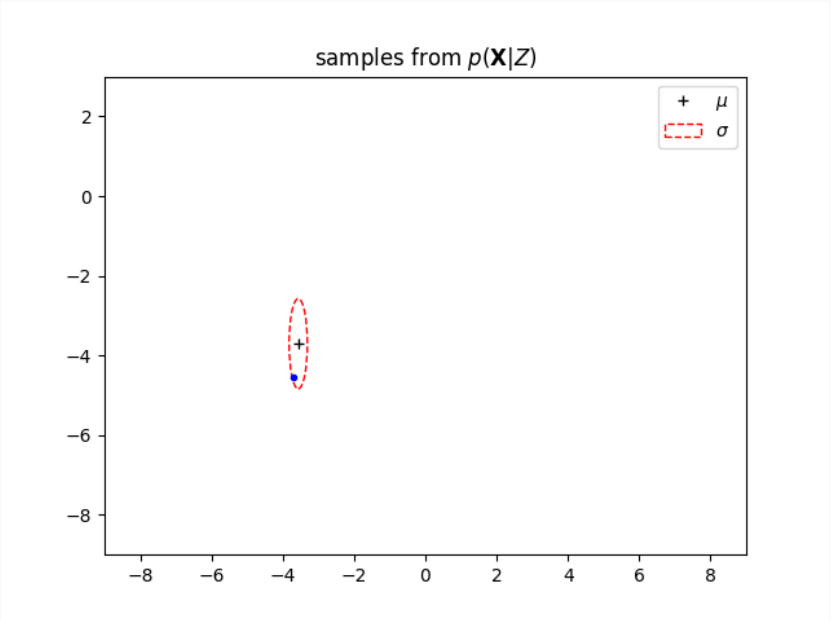
$$p(\boldsymbol{X}) = \overbrace{\int \underbrace{p(\boldsymbol{X} \mid \boldsymbol{z})}_{\textbf{decoder}} \quad \underbrace{p(\boldsymbol{z})}_{\text{white Gauss.}} \quad \mathrm{d}\boldsymbol{z}}^{\boldsymbol{Z} \text{ marginalized from } p(\boldsymbol{X},\boldsymbol{Z})}$$



Baby example of a VAE generative model, with 1d latent code $Z$ and 2d data $\boldsymbol{X}$. I have learned the model on some toy data (we'll see how).

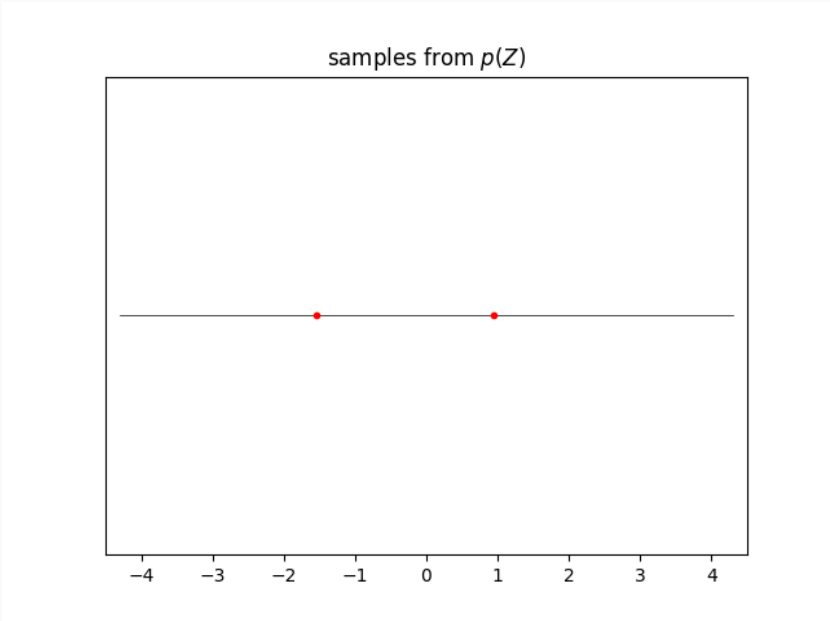
## 1 sample

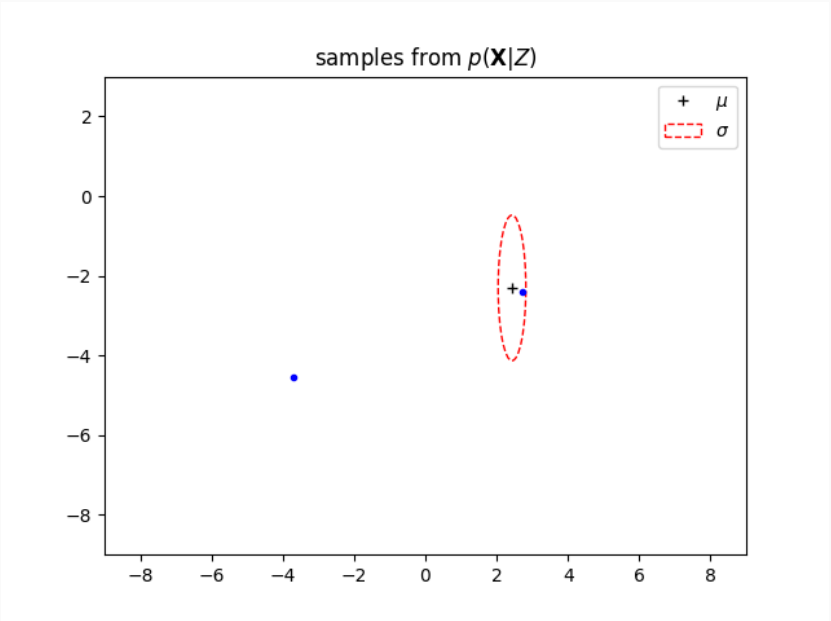


Baby example of a VAE generative model, with 1d latent code $Z$ and 2d data $\boldsymbol{X}$. I have learned the model on some toy data (we'll see how).

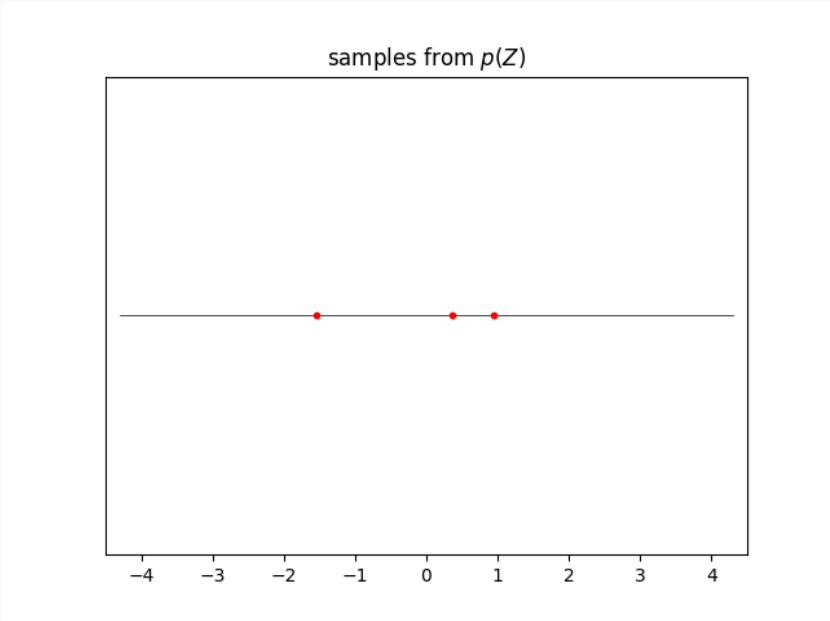
## 2 samples

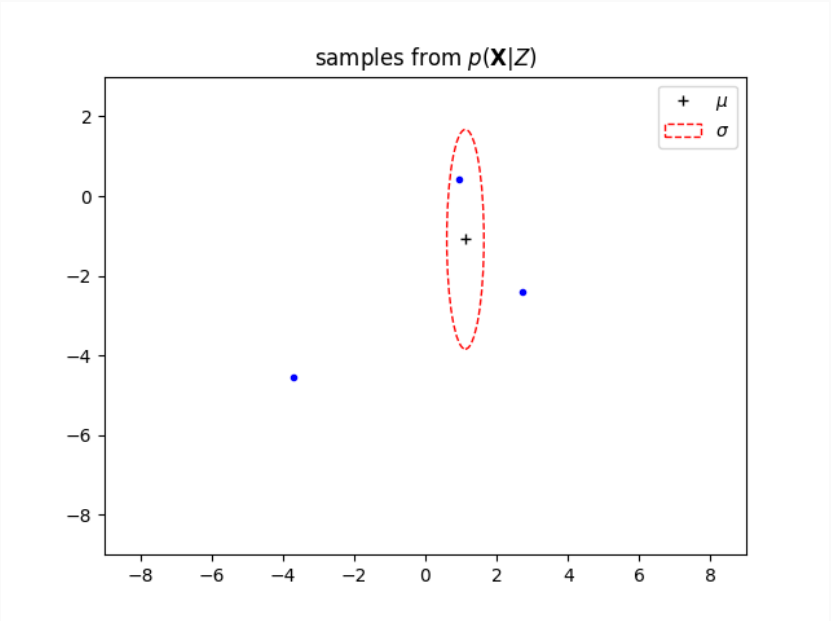


Baby example of a VAE generative model, with 1d latent code $Z$ and 2d data $\boldsymbol{X}$. I have learned the model on some toy data (we'll see how).

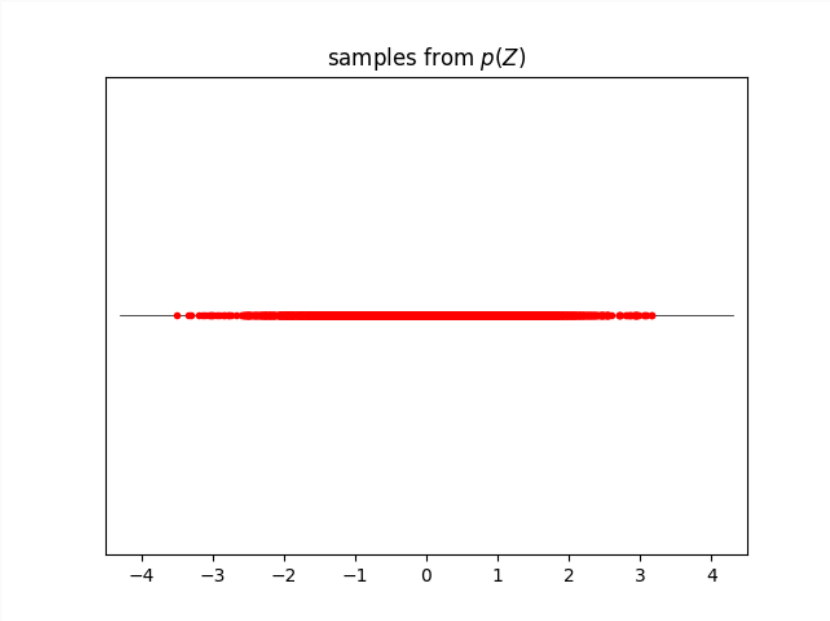
## 3 samples

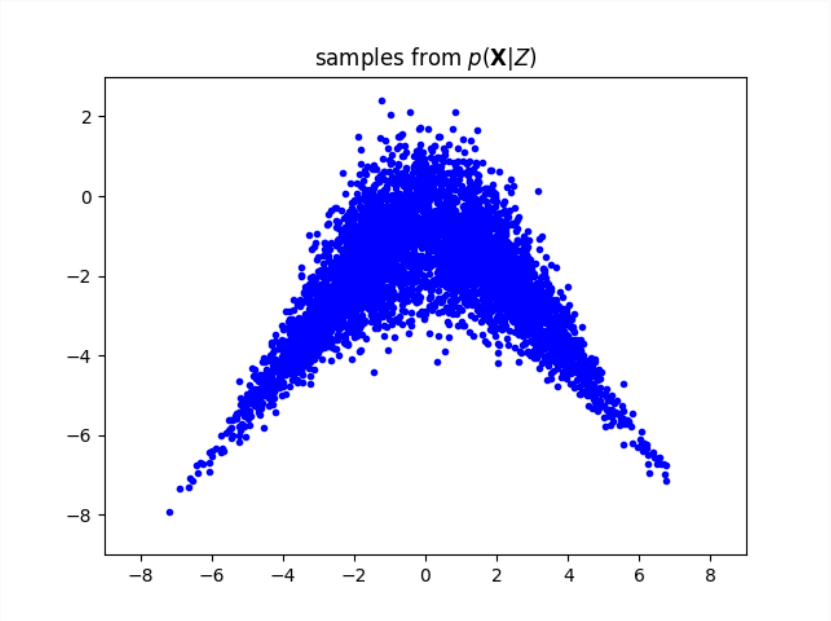


Baby example of a VAE generative model, with 1d latent code $Z$ and 2d data $\boldsymbol{X}$. I have learned the model on some toy data (we'll see how).

**5000 samples**



# Learning VAEs

- we have a well-defined VAE density:

$$p_\theta(\boldsymbol{X}) = \overbrace{\int p_\theta(\boldsymbol{X} \mid \boldsymbol{z})\, p(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z}}^{\boldsymbol{Z} \text{ marginalized from } p(\boldsymbol{X},\boldsymbol{Z})}$$

- here, let $\boldsymbol{\theta}$ denote the parameters of the model
- given data $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, we ideally would want to learn the model by maximizing the log-likelihood:

$$\max_{\boldsymbol{\theta}} \; \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p_\theta(\boldsymbol{x}_i)$$

- however, $p_\theta(\boldsymbol{x}_i)$ contains a high-dimensional integral over an entire neural network, and is hard to compute 

# Evidence Lower Bound

- an effective way to learn the model is by maximizing a **lower bound of** $\mathcal{L}(\theta)$
- Let $q_\phi(\boldsymbol{Z} \mid \boldsymbol{x})$ be **any** distribution over latent code $\boldsymbol{Z}$ conditional on data sample $\boldsymbol{x}$, parametrized by some $\phi$. Then

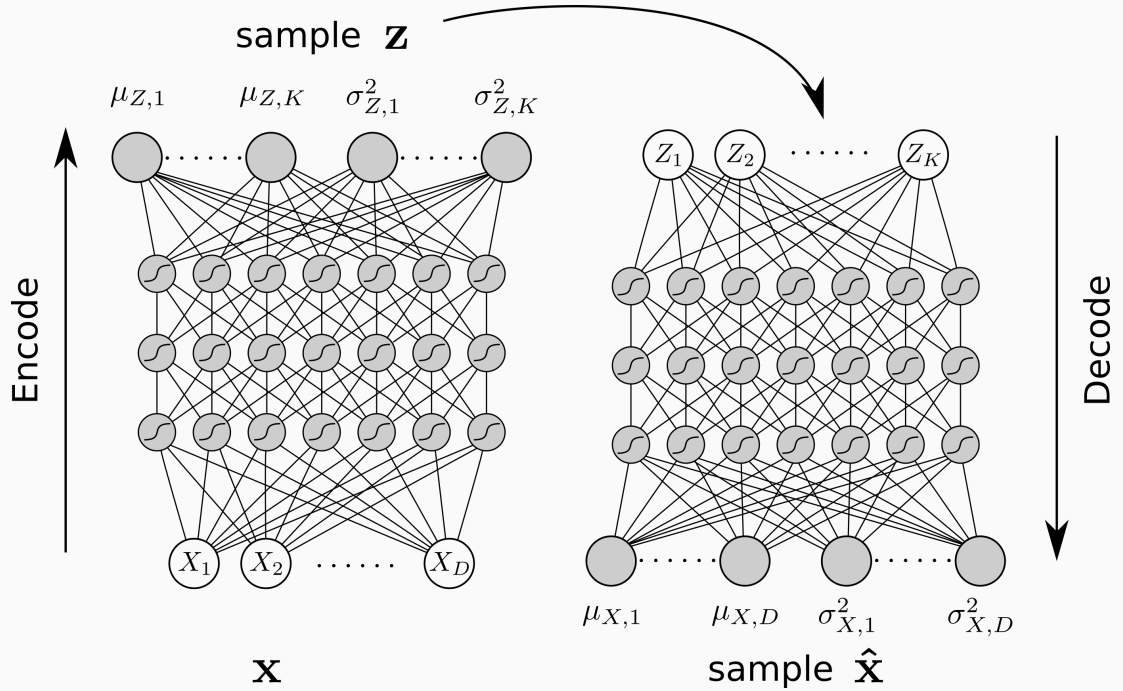
$$ELBO(\theta, \phi) = \sum_{i=1}^{N} \left[ \overbrace{\mathbb{E}_{q_\phi(\boldsymbol{Z}_i \mid \boldsymbol{x}_i)} \left[ \log p_\theta(\boldsymbol{x}_i, \boldsymbol{Z}_i) \right]}^{\text{exp. complete log-likelihood}} + \overbrace{H[q_\phi(\boldsymbol{Z}_i \mid \boldsymbol{x}_i)]}^{\text{entropy of } q_\phi} \right] \leq \mathcal{L}(\theta)$$

- $\mathcal{L}(\theta)$ is often called the **evidence**, hence the lower bound has the famous name **evidence lower bound** (ELBO)
- the lower bound becomes **tight** if $q_\phi(\boldsymbol{Z}_i \mid \boldsymbol{x}_i) = p_\theta(\boldsymbol{Z}_i \mid \boldsymbol{x}_i)$, i.e. when $q_\phi$ is the exact posterior over latent codes under model $\theta$

# VAE Training

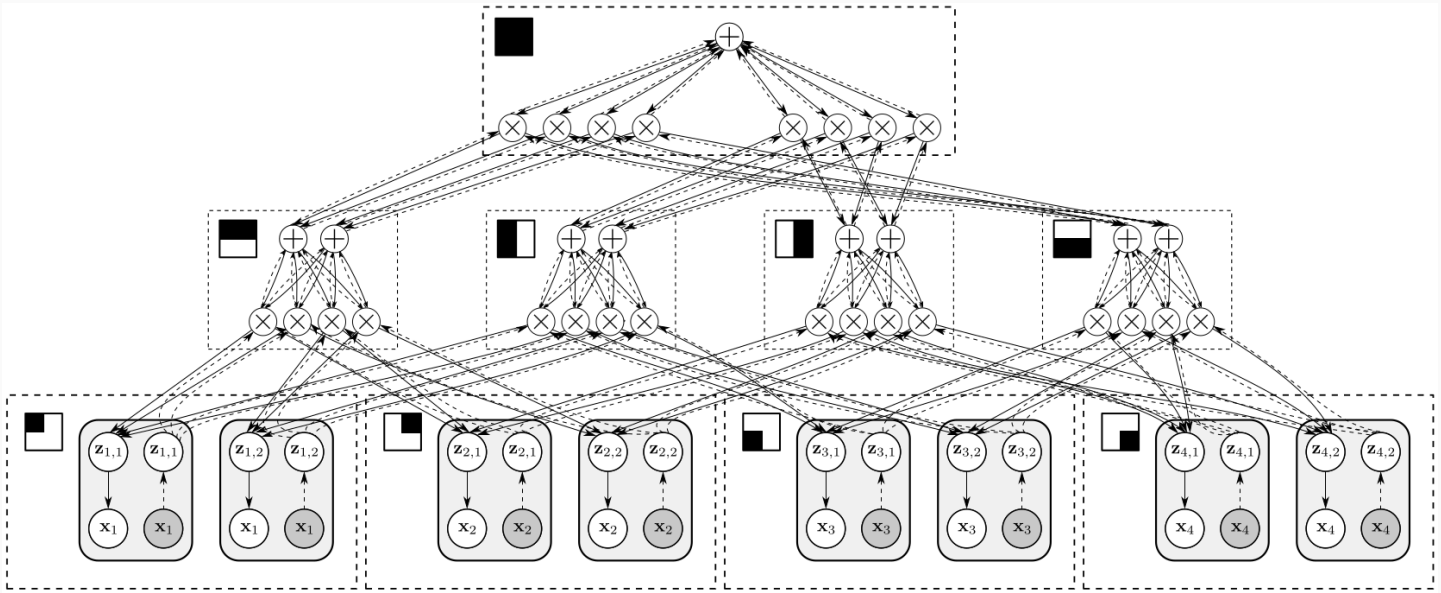
- $q_\phi(\boldsymbol{Z} \mid \boldsymbol{x})$ is usually a neural net with parameters $\phi$ (**encoder**)
- $ELBO(\theta, \phi)$ is maximized w.r.t. both
  - $\theta$ (improve model) and
  - $\phi$ (improve bound)

  using **stochastic mini batch training**
- i.e., learning by maximizing a **noisy lower bound of** $\mathcal{L}(\theta)$



# PC with VAE Leaves (VAEPC) [Tan and Peharz 2019]

- we use now **entire VAE distributions as PC leaves** 
- the VAE model is a well-defined density, so this is "allowed"
- Idea: each VAE is modelling a smaller scope and has a low-dimensional latent code
- many low-dimensional problems should—intuitively—be easier than one high-dimensional problem.



# How to learn VAEPCs?

- VAEPCs are intractable, since they use intractable leaves
- thus, we don't have a density/likelihood for the leaves
- but, we could use an encoder for each leave and come up with noisy lower bound for each leaf
- this is indeed a well-justified objective:

**ELBO Estimate (Tan & Peharz, Theorem 2)**

When replacing exact input distributions with the usual noisy ELBO estimates (VAE, importance-weighted VAE), the VAEPC output is a noisy ELBO estimate for the whole circuit.

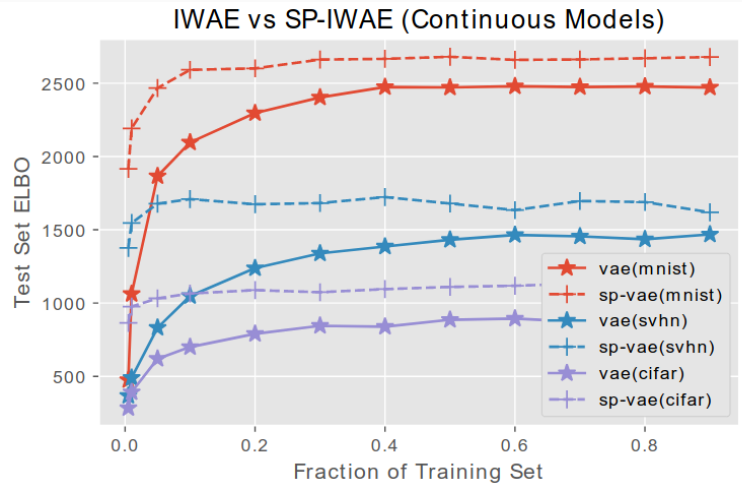
- gradient is simply computed with backprop, through the entire PC and input VAEs, including their encoders!

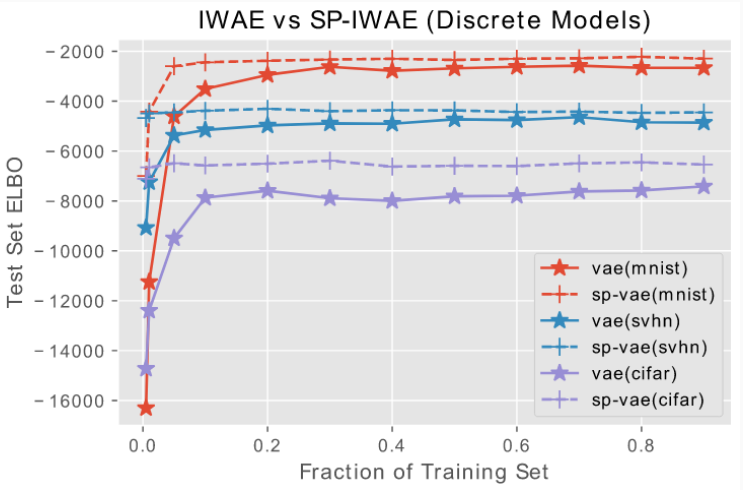
# VAEPC* vs. VAEs

Tan and Peharz, 2019

*Table 1.* Performance on test set, 5000-sample IWAE ELBO

| | Continuous | | | Discrete | | |
|---|---|---|---|---|---|---|
| | mnist | svhn | cifar | mnist | svhn | cifar |
| SPVAE | **2819** | **1936** | **1283** | **-1532** | **-3891** | **-5543** |
| VAE | 2598 | 1442 | 896 | -2351 | -4965 | -7200 |
| Conv-SPVAE | 2702 | **2101** | **1397** | **-927** | **-3666** | **-4562** |
| Conv-VAE | **2907** | 1896 | 1191 | -2099 | -4115 | -6752 |



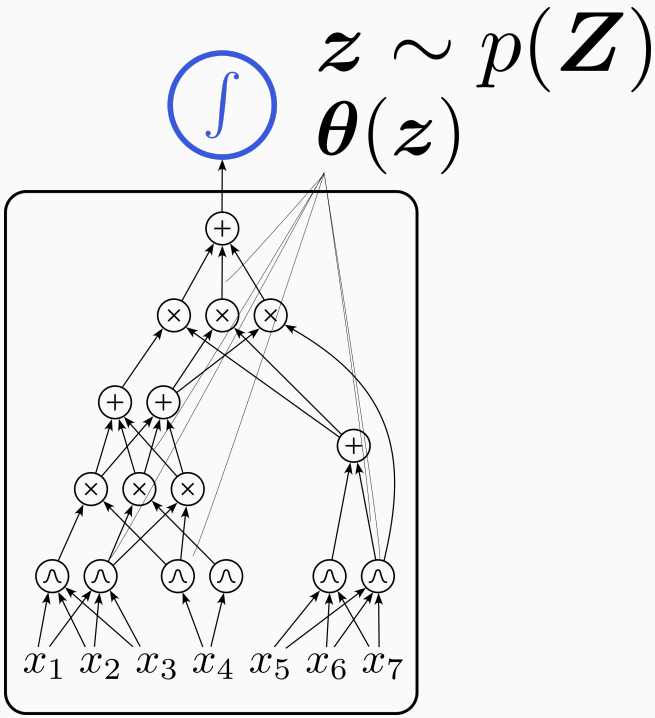

*here VAEPC is called "SPVAE"

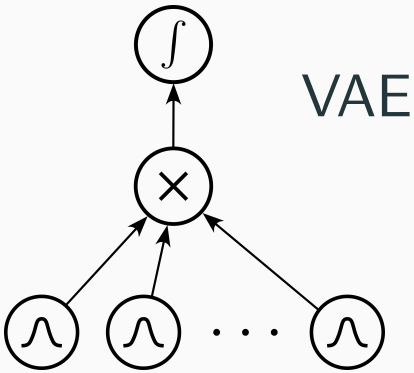
# Probabilistic Integral Circuits

- absorb continuous mixtures into PC with **integral nodes**
- formally, we associate a continuous latent vector $\boldsymbol{Z}$ with each integral node
- the parameters $\boldsymbol{\theta}$ of the PC structure below depend on $\boldsymbol{Z}$ in a non-linear way (neural net)
- the integral node computes

$$p_{int}(\boldsymbol{X}) = \int \overbrace{p(\boldsymbol{X} \mid \boldsymbol{\theta}(\boldsymbol{z}))}^{\text{PC}} \, p(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z}$$

- this yields **probabilistic integral circuits (PICs)**



VAE as PIC

# Quadrature PCs (QPCs)

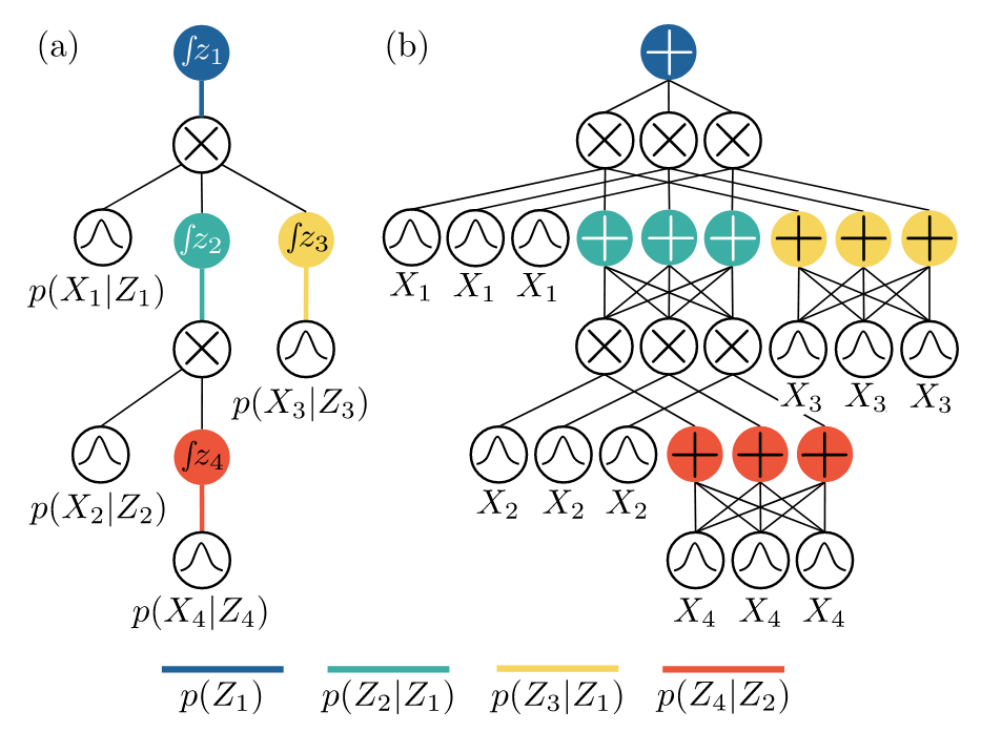
- PICs obviously improve the expressivity of PCs, but they contain an intractable integral over a non-linear function
- however, **low-dimensional** integrals (say, up to 5 dimensions) are pretty ok to solve using **numerical integration**
- in particular, **quadrature rules** like **Gaussian quadrature**, **trapezoidal rule**, **Simpson's rule**, **quasi Monte Carlo** have all the form

$$\int f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \sum_{i=1}^{N} w_i\, f(\boldsymbol{x}_i)$$

- this looks a lot like a sum node 
- quadrature rules converge to the exact integral for $N \to \infty$
- their error can be bounded under certain assumptions on the integrand

# Quadrature PCs (QPCs) cont'd

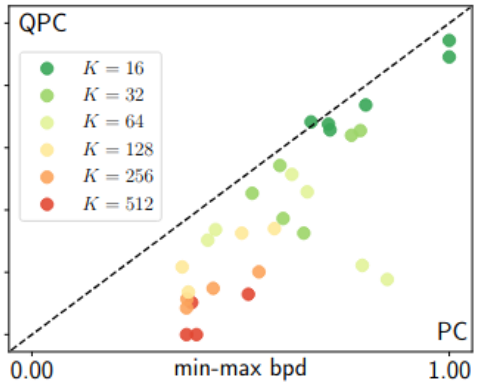
- with suitable conditional independence assumptions between latent RVs, we can perform "local quadratures"
- such **quadrature PC** (QPC) approximate PICs arbitrarily well
- the sum weights in a QPC are not free parameters, but given by the integrand, i.e. indirectly via the neural net



# QPC in MNIST Family

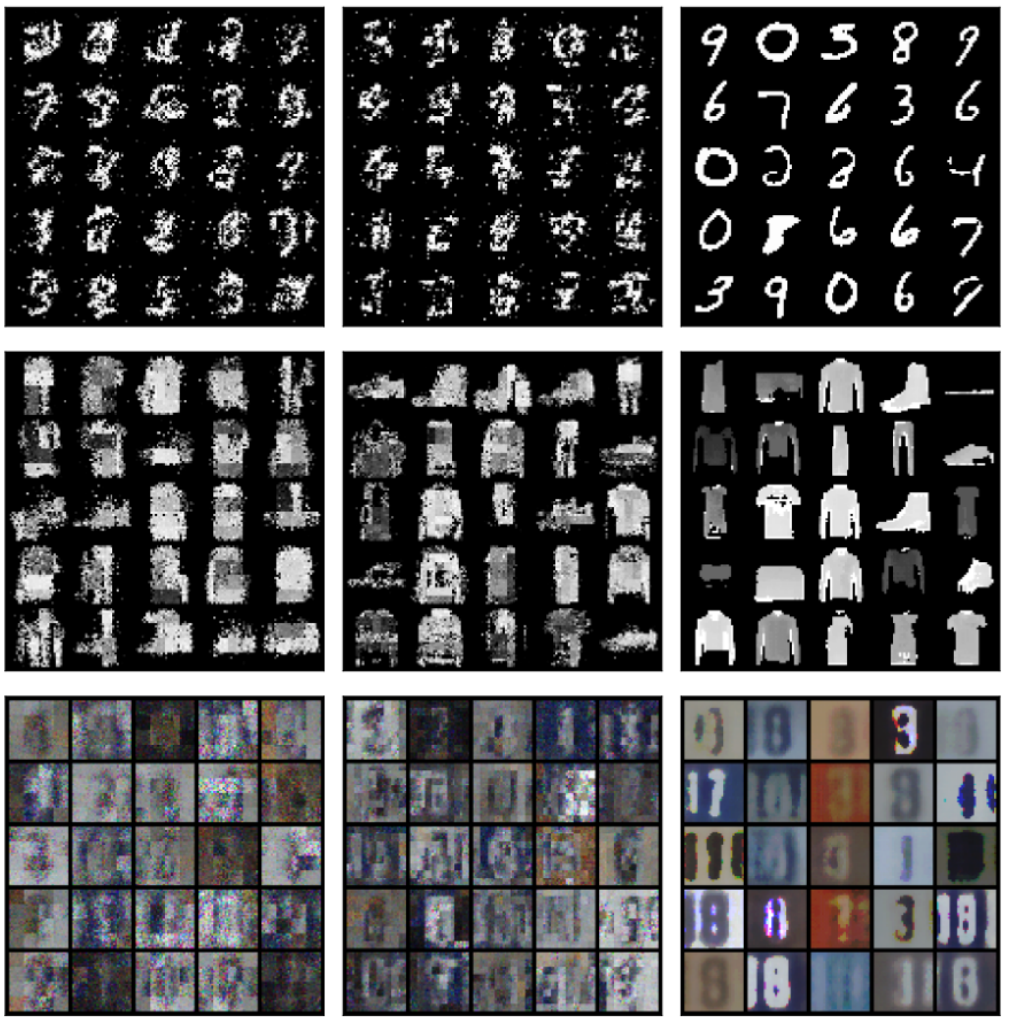
Gala et al., 2024

| | QPC | PC | Sp-PC | HCLT | RAT | IDF | BitS | BBans | McB |
|---|---|---|---|---|---|---|---|---|---|
| MNIST | **1.11** | 1.17 | 1.14 | 1.21 | 1.67 | 1.90 | 1.27 | 1.39 | 1.98 |
| F-MNIST | **3.16** | 3.32 | 3.27 | 3.34 | 4.29 | 3.47 | 3.28 | 3.66 | 3.72 |
| EMN-MN | 1.55 | 1.64 | **1.52** | 1.70 | 2.56 | 2.07 | 1.88 | 2.04 | 2.19 |
| EMN-LE | **1.54** | 1.62 | 1.58 | 1.75 | 2.73 | 1.95 | 1.84 | 2.26 | 3.12 |
| EMN-BA | **1.59** | 1.66 | 1.60 | 1.78 | 2.78 | 2.15 | 1.96 | 2.23 | 2.88 |
| EMN-BY | 1.53 | **1.47** | 1.54 | 1.73 | 2.72 | 1.98 | 1.87 | 2.23 | 3.14 |



# Image Samples

Correia et al., 2023



Left and center columns: samples from EiNets.

Right column: samples from a simple QPC.

- **main stream today:** "use beefy expressive models, how cares about exact inference?"
- however, it is good to **have your reasoning machine intact**
- when something can be solved tractably, why go to intransparent approximations?
- **hybrid inference**: you can mix exact and approximate inference
- PCs are a good tool to have under your belt and are not a standalone tool