Divide, Conquer, and Combine: a New Inference Strategy for Probabilistic Programs with Stochastic Support

Yuan Zhou, Hongseok Yang, Yee Whye Teh, Tom Rainforth Proceedings of the 37th International Conference on Machine Learning, PMLR 119:11534-11545, 2020.

Presented by: Bianca Cosma and Margot Pauëlsen | 25.09.2023

TU Delft

Introduction

- Stochastic variables lead to variable dimensionality
- Model declaration is easy, (automated) inference is hard
- **Goal of the paper:** provide stochastic support to wide range of models

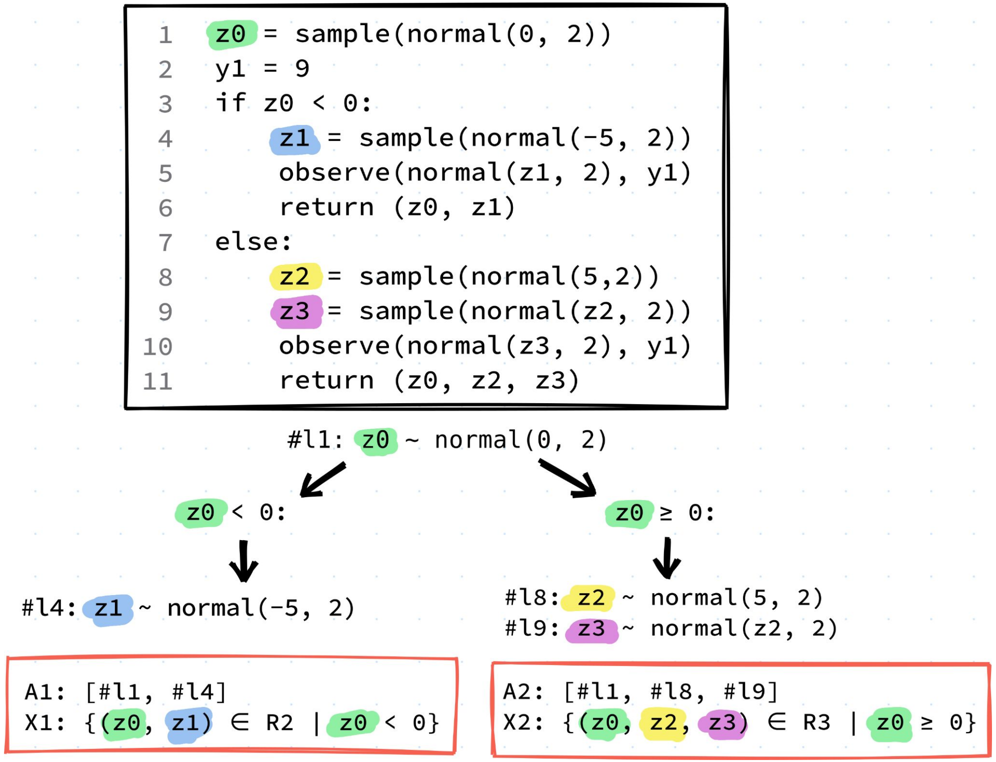


Figure adapted from Zhou et al.

Notation

Sample

The set of **random variables** $\{x_i\}_{i=1}^{n_x} = (x_1, \ldots, x_{nx})$,

where x_i has **address** a_i , **input (history)** η_i and **density (prior)** $f_{a_i}(x_i|\eta_i)$

Observe

The set of **observed values** $\{y_j\}_{j=1}^{n_y} = (y_1, \ldots, y_{n_y})$,

where y_j has **address** b_j , **input** ϕ_j and **density (likelihood)** $g_{b_j}(y_j|\phi_j)$

Notation (continued)

The **posterior p(x | y)** can be expressed as $\pi(x) = \gamma(x)/Z$,

where

(prior) (likelihood)

$$\gamma(x) := \prod_{i=1}^{n_x} f_{a_i}(x_i|\eta_i) \prod_{j=1}^{n_y} g_{b_j}(y_j|\phi_j),$$

joint density p(x, y)

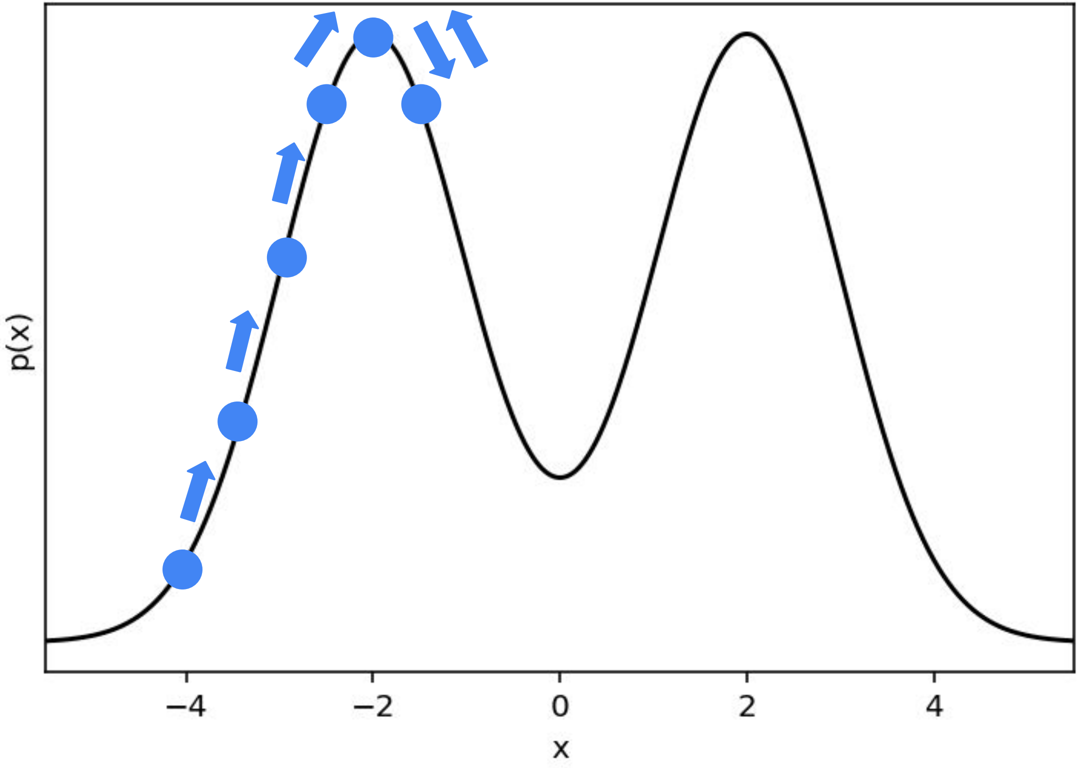
$$Z := \int \prod_{i=1}^{n_x} f_{a_i}(x_i|\eta_i) \prod_{j=1}^{n_y} g_{b_j}(y_j|\phi_j) dx_{1:n_x},$$

marginal likelihood p(y)
(normalizing constant)

Problems with other inference methods

- Rejection and importance sampling suffer ‘curse of dimensionality’
- Variational inference
    - needs specific knowledge, so at odds with automation
    - model often approximated by model with fixed support
- MCMC needs kernel to switch between configurations
    - Posterior may vary across configurations
    - It is difficult to switch from high density region to new configuration

Problems with other inference methods

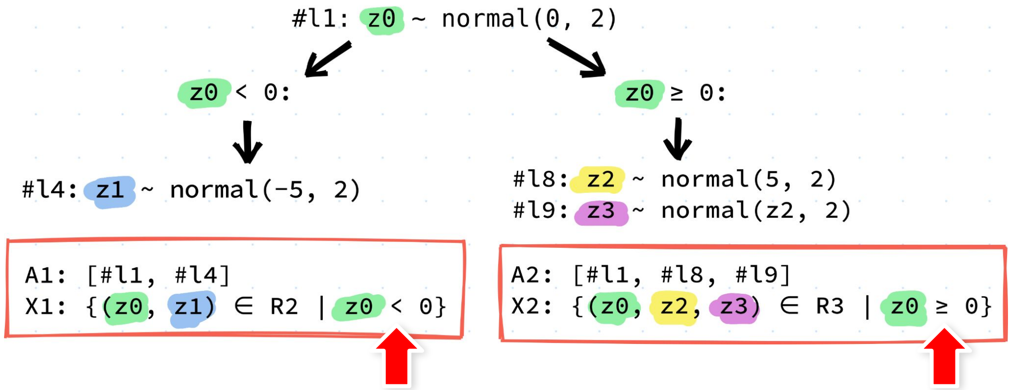


Multimodal distribution image: https://emcee.readthedocs.io/en/stable/tutorials/moves/

Step 1: Divide

- Divide program in straight-line sub-programs (SLPs)
- **The individual programs are disjoint (independent)**

```
z0 = sample(normal(0, 2))
y1 = 9
if z0 < 0:
    z1 = sample(normal(-5, 2))
    observe(normal(z1, 2), y1)
    return (z0, z1)
else:
    z2 = sample(normal(5,2))
    z3 = sample(normal(z2, 2))
    observe(normal(z3, 2), y1)
    return (z0, z2, z3)
```



Step 2: Conquer

For each SLP $\mathbf{k}$, we estimate, using **<u>any conventional inference approach</u>**:

The posterior:	**The marginal likelihood:** (evidence)
$\pi_k(x)$	Z_k
p(x|y)	**p(y)**

Step 3: Combine

joint distributions p(x, y)for all SLPs 1:K

$$\pi(x) = \frac{\sum_{k=1}^{K} \gamma_k(x)}{\sum_{k=1}^{K} Z_k}$$

we want the posterior p(x|y) for the entire program

marginal likelihoods p(y) for all SLPs 1:K

We can sum up disjoint events to get the joint probability

Step 3: Combine (continued)

Why can we just sum the independent SLPs?

for the joint distribution:

$$\gamma(x) = \sum_{k=1}^{K} \gamma_k(x)$$

for the marginal likelihood:

$$Z = \sum_{k=1}^{K} Z_k$$

```
z0 = sample(normal(0, 2))
y1 = 9
if z0 < 0:
    z1 = sample(normal(-5, 2))
    observe(normal(z1, 2), y1)
    return (z0, z1)
else:
    z2 = sample(normal(5,2))
    z3 = sample(normal(z2, 2))
    observe(normal(z3, 2), y1)
    return (z0, z2, z3)
```

```
#l1: z0 ~ normal(0, 2)

z0 < 0:
#l4: z1 ~ normal(-5, 2)

z0 ≥ 0:
#l8: z2 ~ normal(5, 2)
#l9: z3 ~ normal(z2, 2)
```

```
A1: [#l1, #l4]
X1: {(z0, z1) ∈ R2 | z0 < 0}
```

```
A2: [#l1, #l8, #l9]
X2: {(z0, z2, z3) ∈ R3 | z0 ≥ 0}
```

Because the program supports are <u>disjoint</u> (and we can sum up disjoint events to get the joint probability)

How do we find SLPs?

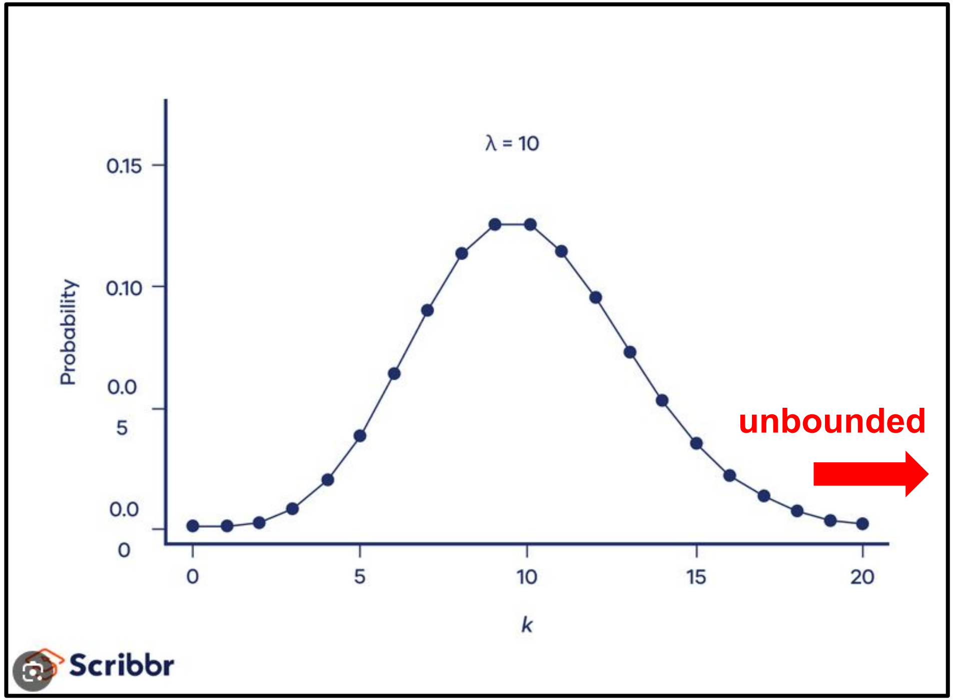
Static analysis doesn't always work: sometimes there's no upper bound on the number of code paths

$K \sim \text{Poisson}(10)$

for $k \in K$:

$\quad w \sim N(\mu_k, \Sigma_k)$

...



Implementation details: finding SLPs at inference time

1. Run the program a number of times to generate an initial set of SLPs

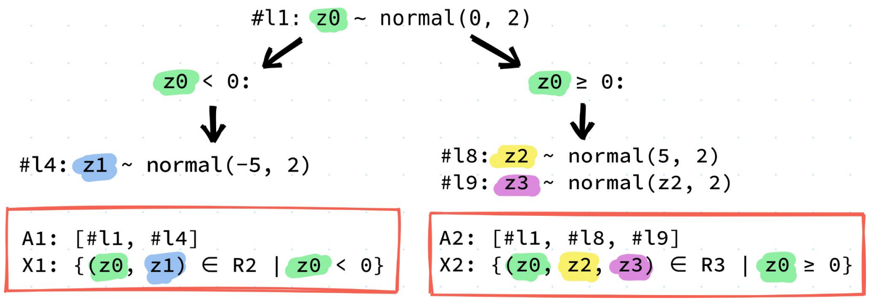
```
Initial set: S = {[#l1, #l4]}
```

which one?

2. At each iteration:
   a. Perform local inference on an SLP (for example, MCMC)
   b. If we find another path, we update our set

```
Assume on #l1 we sample z0 = 0.4 > 0:
  -  so we jump to #l8
  -  we found another path
  -  update:
     S = {[#l1, #l4], [#l1, #l8, #l9]}
```

```
1   z0 = sample(normal(0, 2))
2   y1 = 9
3   if z0 < 0:
4       z1 = sample(normal(-5, 2))
5       observe(normal(z1, 2), y1)
6       return (z0, z1)
7   else:
8       z2 = sample(normal(5,2))
9       z3 = sample(normal(z2, 2))
10      observe(normal(z3, 2), y1)
11      return (z0, z2, z3)
```



Implementation details: resource allocation

At each iteration, how do we choose an SLP to do local inference on?

Pick the one with the highest utility:

inversely proportional to the number of
times we have already performed inference

$$U_k := \boxed{\frac{1}{S_k}} \left(\boxed{\frac{(1-\delta)\hat{\tau}_k}{\max_k\{\hat{\tau}_k\}}} + \boxed{\frac{\delta \hat{p}_k}{\max_k\{\hat{p}_k\}}} + \frac{\beta \log \sum_k S_k}{\sqrt{S_k}} \right)$$

exploitation

exploration

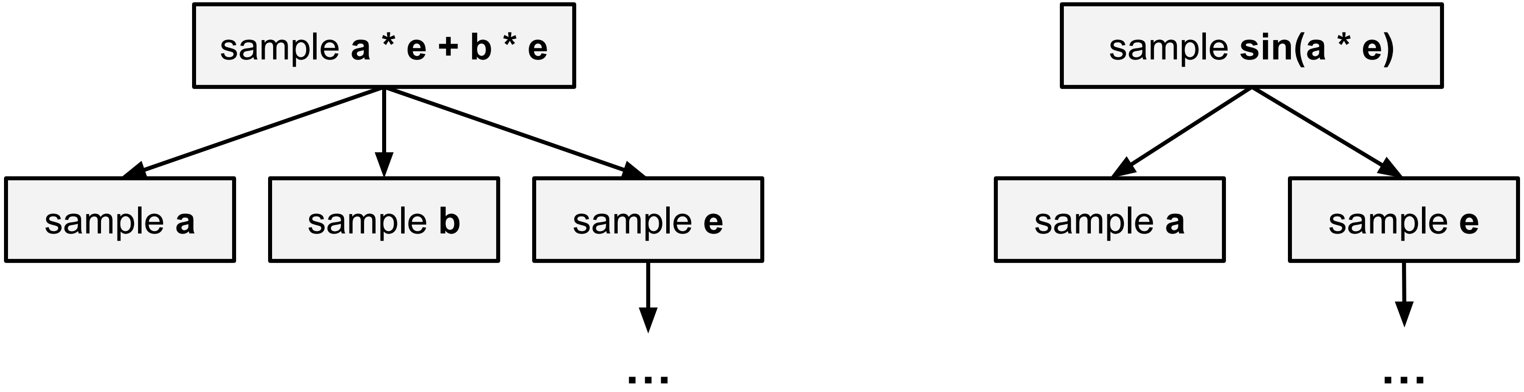
utility function for an SLP k

proportional to Z_k
(likelihood of the SLP)

Experiments

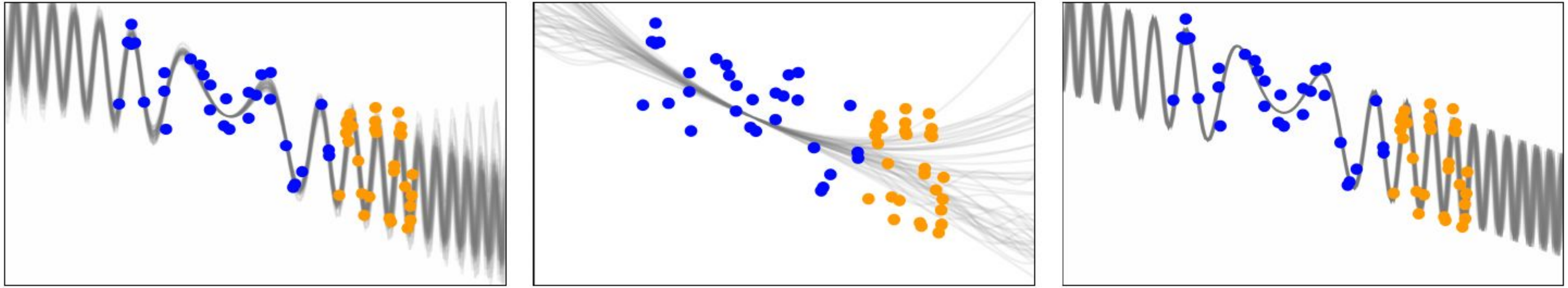
Function induction: we want to estimate the function that generated our data

- the set of rules: $e \rightarrow \{x \mid x^2 \mid \sin(a*e) \mid a*e+b*e\}$
- why does it make sense to apply “Divide, Conquer and Combine”?



Experiments

Function induction: we want to estimate the function that generated our data



(a) DCC (ours) | (b) IS | (c) RMH

(Divide, Conquer and Combine) | (importance sampling) | (MCMC: random-walk Metropolis Hastings)