Exact Inference for Discrete Probabilistic Programs via Generating Functions

Fabian Zaiser, Luke Ong

University of Oxford

LAFI workshop @ POPL, 2023-01-15

An Inference Problem

- Your coworker gets 10 calls per week on average.
- 20% of calls are scams.
- At the end of the week, your coworker got only 1 scam call.

An Inference Problem

- Your coworker gets 10 calls per week on average.
- 20% of calls are scams.
- At the end of the week, your coworker got only 1 scam call.

How many calls did they get?

An Inference Problem

- Your coworker gets 10 calls per week on average.
- 20% of calls are scams.
- At the end of the week, your coworker got only 1 scam call.

How many calls did they get?

Probabilistic program

$$X \sim \text{Poisson}(10)$$
$$Y \sim \text{Binomial}(X, 0.2)$$
$$\text{observe } Y = 1$$

Tools for Exact Inference

$$
\begin{aligned}
&X \sim \text{Poisson}(10) \\
&Y \sim \text{Binomial}(X, 0.2) \\
&\text{observe } Y = 1
\end{aligned}
$$

Tools for Exact Inference

$$X \sim \text{Poisson}(10)$$
$$Y \sim \text{Binomial}(X, 0.2)$$
$$\text{observe } Y = 1$$

Dice [Holtzen et al. 2020]

✗ Only supports finite discrete distributions.

Tools for Exact Inference

$$X \sim \text{Poisson}(10)$$
$$Y \sim \text{Binomial}(X, 0.2)$$
$$\text{observe}\, Y = 1$$

Dice [Holtzen et al. 2020]

✗ Only supports finite discrete distributions.

SPPL [Saad et al. 2021]

✗ Parameters of distributions must have finite support.

Tools for Exact Inference

$$\begin{aligned} X &\sim \text{Poisson}(10) \\ Y &\sim \text{Binomial}(X, 0.2) \\ \text{observe } Y &= 1 \end{aligned}$$

Dice [Holtzen et al. 2020]

✗ Only supports finite discrete distributions.

SPPL [Saad et al. 2021]

✗ Parameters of distributions must have finite support.

PSI [Gehr et al. 2016]

✗ Outputs a symbolic expression involving infinite sums.

Infinite Support

$$X \sim \text{Poisson}(10)$$
$$Y \sim \text{Binomial}(X, 0.2)$$

Infinite Support

$$X \sim \mathsf{Poisson}(10)$$
$$Y \sim \mathsf{Binomial}(X, 0.2)$$

$$\mathbb{P}[Y = 9] = \sum_{x=0}^{\infty} \mathbb{P}[X = x]\mathbb{P}[Y = 9 \mid X = x]$$

Infinite Support

$$X \sim \text{Poisson}(10)$$
$$Y \sim \text{Binomial}(X, 0.2)$$

$$\mathbb{P}[Y = 9] = \sum_{x=0}^{\infty} \mathbb{P}[X = x]\mathbb{P}[Y = 9 \mid X = x]$$

Not computable exactly using probability *mass* functions!

A probabilistic language

Fixed number of variables $X_1, \ldots, X_n$, taking values in $\mathbb{N}$.

A probabilistic language

Fixed number of variables $X_1, \ldots, X_n$, taking values in $\mathbb{N}$.

- **Affine transform**: $X_i := aX_j + bX_k + c$
  (where $a, b, c \in \mathbb{N}$)

A probabilistic language

Fixed number of variables $X_1, \ldots, X_n$, taking values in $\mathbb{N}$.

- **Affine transform**: $X_i := aX_j + bX_k + c$
  (where $a, b, c \in \mathbb{N}$)
- **Branching**: if $X_i = c \, \{P_1\}$ else $\{P_2\}$

A probabilistic language

Fixed number of variables $X_1, \ldots, X_n$, taking values in $\mathbb{N}$.

- **Affine transform**: $X_i := aX_j + bX_k + c$
  (where $a, b, c \in \mathbb{N}$)
- **Branching**: if $X_i = c \, \{P_1\}$ else $\{P_2\}$
- **Sampling**: $X_i \sim \mathcal{D}$

A probabilistic language

Fixed number of variables $X_1, \dots, X_n$, taking values in $\mathbb{N}$.

- **Affine transform**: $X_i := aX_j + bX_k + c$ (where $a, b, c \in \mathbb{N}$)
- **Branching**: if $X_i = c\,\{P_1\}$ else $\{P_2\}$
- **Sampling**: $X_i \sim \mathcal{D}$

$$\mathcal{D} \in \{\text{Bernoulli}(p), \text{Binomial}(X_k, p), \text{Geometric}(p), \text{Poisson}(\lambda \cdot X_k)\}.$$

Klinkenberg et al. 2020

A probabilistic language

Fixed number of variables $X_1, \dots, X_n$, taking values in $\mathbb{N}$.

- **Affine transform**: $X_i := aX_j + bX_k + c$ (where $a, b, c \in \mathbb{N}$)
- **Branching**: if $X_i = c \, \{P_1\}$ else $\{P_2\}$
- **Sampling**: $X_i \sim \mathcal{D}$

$$\mathcal{D} \in \{\text{Bernoulli}(p), \text{Binomial}(X_k, p), \text{Geometric}(p), \text{Poisson}(\lambda \cdot X_k)\}.$$

Klinkenberg et al. 2020

- **Conditioning**: observe $X_i = c$
- **Nested inference**: normalize $\{P\}$

Our contribution

A probabilistic language

Fixed number of variables $X_1, \ldots, X_n$, taking values in $\mathbb{N}$.

- **Affine transform**: $X_i := aX_j + bX_k + c$ (where $a, b, c \in \mathbb{N}$)
- **Branching**: if $X_i = c$ $\{P_1\}$ else $\{P_2\}$
- **Sampling**: $X_i \sim \mathcal{D}$

$$\mathcal{D} \in \{\text{Bernoulli}(p), \text{Binomial}(X_k, p), \text{Geometric}(p), \text{Poisson}(\lambda \cdot X_k)\}.$$

Klinkenberg et al. 2020

- **Conditioning**: observe $X_i = c$
- **Nested inference**: normalize $\{P\}$

Our contribution

⇝ can express **real-world models**, e.g. population dynamics & change point models

Probability Generating Functions (PGFs)

Generating function of $X \sim \mathcal{D}$ (supported on $\mathbb{N}$):

Probability Generating Functions (PGFs)

Generating function of $X \sim \mathcal{D}$ (supported on $\mathbb{N}$):

$$\begin{aligned}
\mathsf{pgf}_X(x) &= \mathbb{E}[x^X] \quad \text{(discrete \& continuous } X\text{)} \\
&= \sum_{n=0}^{\infty} p_n x^n \quad \text{(only discrete } X\text{)} \\
&\text{where } p_n = \mathbb{P}[X = n]
\end{aligned}$$

Probability Generating Functions (PGFs)

Generating function of $X \sim \mathcal{D}$ (supported on $\mathbb{N}$):

$$\begin{aligned}\mathsf{pgf}_X(x) &= \mathbb{E}[x^X] \quad \text{(discrete \& continuous } X\text{)}\\ &= \sum_{n=0}^{\infty} p_n x^n \quad \text{(only discrete } X\text{)}\\ &\text{where } p_n = \mathbb{P}[X = n]\end{aligned}$$

This infinite sum can often be expressed in **closed form**!

$\mathcal{D}$	$\mathsf{pgf}_{\mathcal{D}}(x)$
Binomial(n, p)	$(px + 1 - p)^n$
Poisson(λ)	$e^{\lambda(x-1)}$

Generating Function Semantics

Represent the distribution over states as a generating function G.

Generating Function Semantics

Represent the distribution over states as a generating function G.

Transformer semantics: $\underbrace{G}_{\text{distribution before}} \xrightarrow{\text{command } C} \underbrace{[\![C]\!](G)}_{\text{distribution after } C}$

Generating Function Semantics

Represent the distribution over states as a generating function G.

Transformer semantics: $\underbrace{G}_{\text{distribution before}} \xrightarrow{\text{command } C} \underbrace{[\![C]\!](G)}_{\text{distribution after } C}$

Closed form is preserved for our language!

Generating Function Semantics

Represent the distribution over states as a generating function G.

Transformer semantics: $\underbrace{G}_{\text{distribution before}} \xrightarrow{\text{command } C} \underbrace{[\![C]\!](G)}_{\text{distribution after } C}$

Closed form is preserved for our language!

To evaluate G, no computer algebra is needed, just **automatic differentiation**.

Semantics of Conditioning

$$[\![\textsf{observe}\, X = c]\!](G)(x) = \frac{G^{(c)}(0)}{c!} \cdot x^c$$

keeps only the term with x^c in the power series.

Semantics of Conditioning

$$[\![\textsf{observe}\, X = c]\!](G)(x) = \frac{G^{(c)}(0)}{c!} \cdot x^c$$

keeps only the term with x^c in the power series.

Observations are expensive!

$\textsf{observe}\, X = 100$ requires evaluating the 100th derivative!

Extracting information from PGFs

Suppose X has generating function $G(x) = \sum_{n \in \mathbb{N}} p_n x^n$.

Extracting information from PGFs

Suppose X has generating function $G(x) = \sum_{n \in \mathbb{N}} p_n x^n$.

- Then the p_n are the Taylor coefficients at $x = 0$, so

$$P[X = n] = p_n = \frac{G^{(n)}(0)}{n!}.$$

Extracting information from PGFs

Suppose X has generating function $G(x) = \sum_{n \in \mathbb{N}} p_n x^n$.

- Then the p_n are the Taylor coefficients at $x = 0$, so
$$P[X = n] = p_n = \frac{G^{(n)}(0)}{n!}.$$
- Then the expected value is: $\mathbb{E}[X] = G'(1)$.

Extracting information from PGFs

Suppose X has generating function $G(x) = \sum_{n \in \mathbb{N}} p_n x^n$.

- Then the p_n are the Taylor coefficients at $x = 0$, so

$$P[X = n] = p_n = \frac{G^{(n)}(0)}{n!}.$$

- Then the expected value is: $\mathbb{E}[X] = G'(1)$.
- Then the variance and higher moments can be expressed with higher derivatives $G^{(n)}(1)$.

Example

$$\textsf{normalize}\,\{X \sim \textsf{Poisson}(10); Y \sim \textsf{Binomial}(X, 0.2); \textsf{observe}\, Y = 1; \}$$

Example

$$\mathsf{normalize}\,\{X \sim \mathsf{Poisson}(10); Y \sim \mathsf{Binomial}(X, 0.2); \mathsf{observe}\, Y = 1;\}$$

1

Example

$$\text{normalize}\,\{X \sim \text{Poisson}(10); Y \sim \text{Binomial}(X, 0.2); \text{observe}\, Y = 1; \}$$

$$1 \xrightarrow{[\![X \sim Poisson(10)]\!]} \exp(10(x-1))$$

Example

normalize $\{X \sim$ Poisson$(10); Y \sim$ Binomial$(X, 0.2);$ observe $Y = 1; \}$

$$1 \xrightarrow{[\![X \sim Poisson(10)]\!]} \exp(10(x-1))$$

$$\xrightarrow{[\![Y \sim \text{Binomial}(X, 0.2)]\!]} \exp(10(x(0.2y + 0.8) - 1))$$

Example

$$\text{normalize}\,\{X \sim \text{Poisson}(10); Y \sim \text{Binomial}(X, 0.2); \text{observe}\, Y = 1; \}$$

$$1 \xrightarrow{[\![X \sim Poisson(10)]\!]} \exp(10(x-1))$$

$$\xrightarrow{[\![Y \sim \text{Binomial}(X, 0.2)]\!]} \exp(10(x\,(0.2y+0.8)-1))$$

$$\xrightarrow{[\![\text{observe}\, Y = 1]\!]} y \cdot \left(\frac{\partial}{\partial y} \exp(10(x(0.2y+0.8)-1))\right)\Bigg|_{y=0} = 2xye^{8x-10}$$

Example

$$\text{normalize}\,\{X \sim \text{Poisson}(10); Y \sim \text{Binomial}(X, 0.2); \text{observe}\, Y = 1;\}$$

$$1 \xrightarrow{[\![X \sim Poisson(10)]\!]} \exp(10(x-1))$$

$$\xrightarrow{[\![Y \sim \text{Binomial}(X, 0.2)]\!]} \exp(10(x(0.2y+0.8)-1))$$

$$\xrightarrow{[\![\text{observe}\, Y=1]\!]} y \cdot \left(\frac{\partial}{\partial y} \exp(10(x(0.2y+0.8)-1))\right)\Big|_{y=0} = 2xye^{8x-10}$$

$$\xrightarrow{[\![\text{normalize}\,\{\}]\!]} \frac{1}{2e^{-2}} 2xye^{8x-10} = xye^{8x-8}$$

Example

normalize $\{X \sim \text{Poisson}(10); Y \sim \text{Binomial}(X, 0.2); \text{observe}\, Y = 1;\}$

$$1 \xrightarrow{[\![X \sim Poisson(10)]\!]} \exp(10(x-1))$$

$$\xrightarrow{[\![Y \sim \text{Binomial}(X, 0.2)]\!]} \exp(10(x(0.2y + 0.8) - 1))$$

$$\xrightarrow{[\![\text{observe}\, Y = 1]\!]} y \cdot \left(\frac{\partial}{\partial y} \exp(10(x(0.2y+0.8)-1))\right)\Big|_{y=0} = 2xye^{8x-10}$$

$$\xrightarrow{[\![\text{normalize}\, \{\}]\!]} \frac{1}{2e^{-2}} 2xye^{8x-10} = xye^{8x-8}$$

Extracting information:

- $\mathbb{P}[X = 10] = \frac{1}{10!} \frac{\partial^{10}}{\partial x^{10}} xye^{8x-8}\Big|_{x=0, y=1} = \frac{1048576}{2835} e^{-8}$

Example

$$\text{normalize}\,\{X \sim \text{Poisson}(10); Y \sim \text{Binomial}(X, 0.2); \text{observe}\, Y = 1;\}$$

$$1 \xrightarrow{[\![X\sim Poisson(10)]\!]} \exp(10(x-1))$$

$$\xrightarrow{[\![Y\sim\text{Binomial}(X,0.2)]\!]} \exp(10(x\,(0.2y+0.8)-1))$$

$$\xrightarrow{[\![\text{observe}\,Y=1]\!]} y \cdot \left(\frac{\partial}{\partial y}\exp(10(x(0.2y+0.8)-1))\right)\Big|_{y=0} = 2xye^{8x-10}$$

$$\xrightarrow{[\![\text{normalize}\,\{\}]\!]} \frac{1}{2e^{-2}}2xye^{8x-10} = xye^{8x-8}$$

Extracting information:

- $\mathbb{P}[X=10] = \frac{1}{10!}\frac{\partial^{10}}{\partial x^{10}}xye^{8x-8}\Big|_{x=0,y=1} = \frac{1048576}{2835}e^{-8}$
- $\mathbb{E}[X] = \frac{\partial}{\partial x}xye^{8x-8}\Big|_{x=1,y=1} = 9$

Demo (population model)

```
normalize {
  population := 0;
  arrivals ~ Poisson(58.15);
  survivors ~ Binomial(population, 0.2636);
  population := arrivals + survivors;
  observed ~ Binomial(population, 0.2);
  observe observed = 0;

  arrivals ~ Poisson(105.2);
  survivors ~ Binomial(population, 0.2636);
  population := arrivals + survivors;
  observed ~ Binomial(population, 0.2);
  observe observed = 12;

  arrivals ~ Poisson(75.2);
  survivors ~ Binomial(population, 0.2636);
  population := arrivals + survivors;
  observed ~ Binomial(population, 0.2);
  observe observed = 24;

  arrivals ~ Poisson(21.4);
  survivors ~ Binomial(population, 0.2636);
  population := arrivals + survivors;
  observed ~ Binomial(population, 0.2);
  observe observed = 21;
}
return population
```

```
p(68) = 0.028328180265953493
p(69) = 0.023065973853818575
p(70) = 0.01838674214279982
p(71) = 0.01435492147905039
p(72) = 0.01098084809984991
p(73) = 0.008233399343568646
p(74) = 0.006053318560300744
p(75) = 0.004365517352330878
p(76) = 0.0030892768898579935
p(77) = 0.0021458658791949775
p(78) = 0.0014635700766538772
p(79) = 0.0009804499356385623
p(80) = 0.0006453115328326522
p(81) = 0.00041741910372653385
p(82) = 0.0002654341174435854
p(83) = 0.00016597450666935572
p(84) = 0.00010208012245584834
p(85) = 0.00006176856103264198
p(86) = 0.0000367813977215097
p(87) = 0.000021558974254102993
p(88) = 0.000012441382544265904
p(89) = 7.070472990081274e-6
p(90) = 3.95788624737007e-6
p(n) <= 4.693893727897103e-6 for all n >= 91

Total measure:  Z = 1.0000000000000002
Expected value: E = 60.28009872152928
Variance:       V = 38.29566931328145

Time elapsed: 1.3375798s
```

Implementation

Implementation

- Computation of derivatives is the bottleneck.

Implementation

- Computation of derivatives is the bottleneck.
- We avoid computer algebra in favor of automatic differentiation.

Implementation

- Computation of derivatives is the bottleneck.
- We avoid computer algebra in favor of automatic differentiation.
- Existing autodiff frameworks very slow for higher-order derivatives

Implementation

- Computation of derivatives is the bottleneck.
- We avoid computer algebra in favor of automatic differentiation.
- Existing autodiff frameworks very slow for higher-order derivatives
- Manual implementation of autodiff is faster

Implementation

- Computation of derivatives is the bottleneck.
- We avoid computer algebra in favor of automatic differentiation.
- Existing autodiff frameworks very slow for higher-order derivatives
- Manual implementation of autodiff is faster
- Computing directly with Taylor expansions is even better

Limitations

Language features:

- only affine functions
- only comparisons $X = c$ (e.g. no $X = Y$)
- only discrete distributions
- no loops/recursion

Limitations

Language features:

- only affine functions
- only comparisons $X = c$ (e.g. no $X = Y$)
- only discrete distributions
- no loops/recursion

Performance:

- GFs can grow exponentially with constants in the program
- worst-case exponential time

Summary

Generating functions **represent distributions** with infinite support **compactly**.

They are a **powerful tool** for **exact inference** in probabilistic programming.