Differentiable probabilistic programming with Pyro

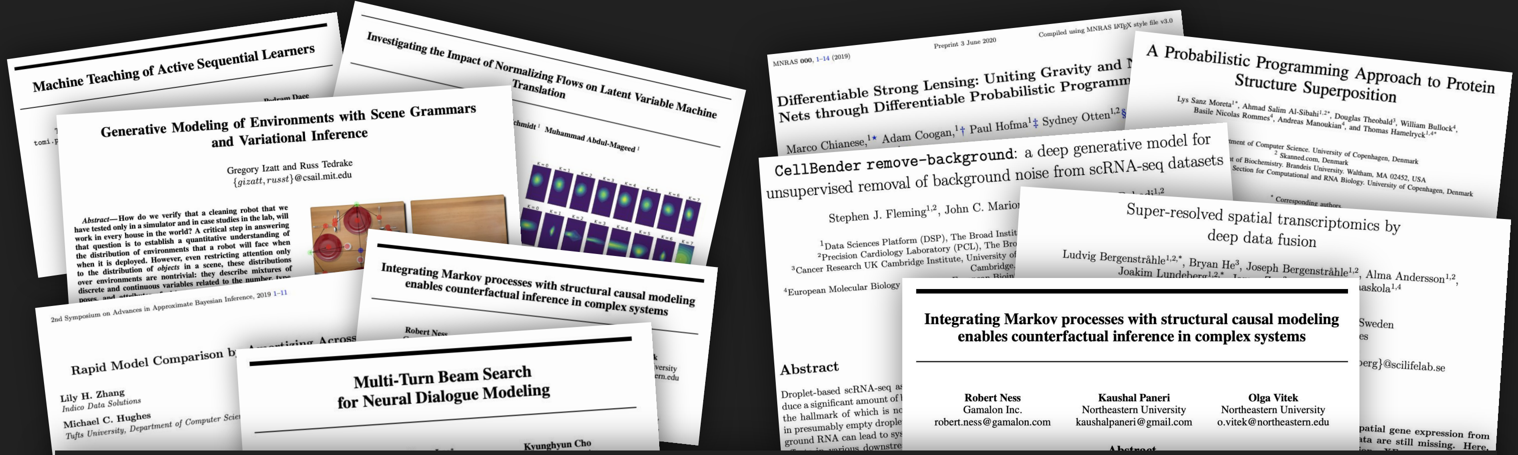
Fritz Obermeyer, Broad Institute
at MODE workshop, 2021-09-08

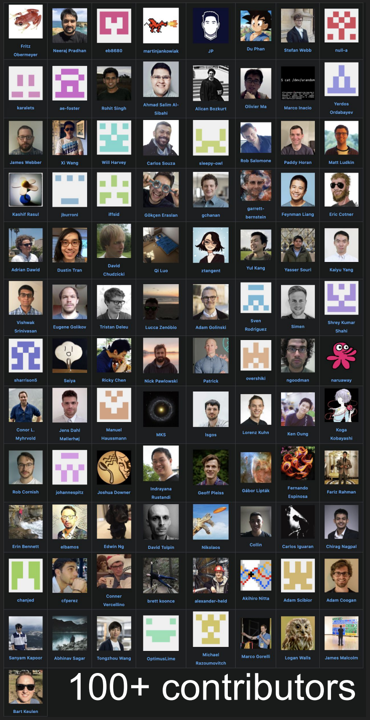
Pyro was open sourced by Uber AI Labs in 2017

adopted by the Linux Foundation in 2019

Python library built on PyTorch / JAX

Catalyzed **ML Research** | Catalyzed **Science**





Corporations: Uber, IBM, Siemens, Apple, UnitedHealth Group, …

Startups: [Robinhood](), [Babylon Health](), [Noodle.ai](), [www.finn.no](), …

Courses: Northeastern, Columbia, UIUC, ...

220+ citations, 800+ github forks, 300+ github repos, 3.8k forum posts, 7 core members

100+ contributors

Overview

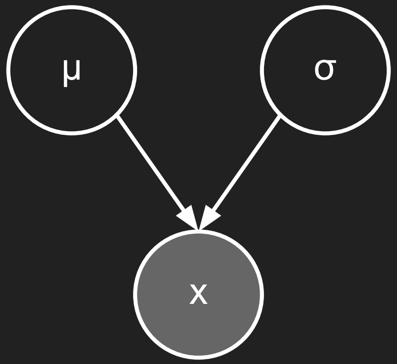
What is a probabilistic model?

What is probabilistic inference?

Intro to Pyro

What is a Probabilistic Model?

$\mu \sim \text{Normal}(0, 10)$
$\sigma \sim \text{LogNormal}(0, 5)$
$x \sim \text{Normal}(\mu, \sigma)$

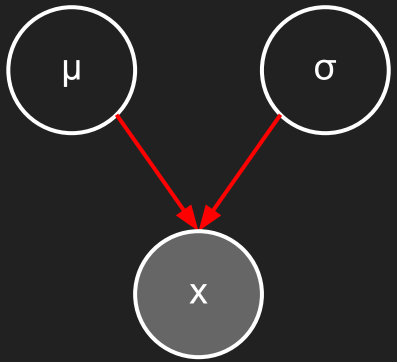


What is a Probabilistic Model?

$\mu \sim \text{Normal}(0, 10)$

$\sigma \sim \text{LogNormal}(0, 5)$

$x \sim \text{Normal}(\mu, \sigma)$



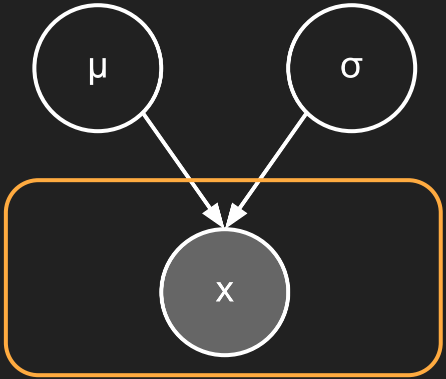
What is a Probabilistic Model?

$\mu \sim \text{Normal}(0, 10)$

$\sigma \sim \text{LogNormal}(0, 5)$

$\boxed{x \sim \text{Normal}(\mu, \sigma)}$

observed data



What is a Probabilistic Model?

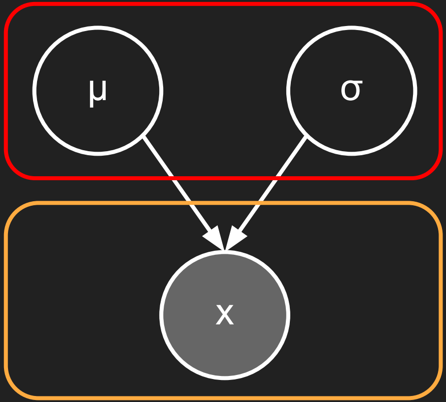
$\mu \sim \text{Normal}(0, 10)$

$\sigma \sim \text{LogNormal}(0, 5)$

$x \sim \text{Normal}(\mu, \sigma)$

latent variables

observed data

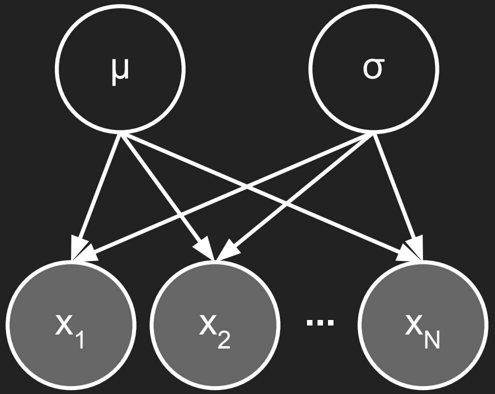


What is a Probabilistic Model?

$\mu \sim \text{Normal}(0, 10)$

$\sigma \sim \text{LogNormal}(0, 5)$

$x_1,\ldots,x_N \overset{iid}{\sim} \text{Normal}(\mu, \sigma)$

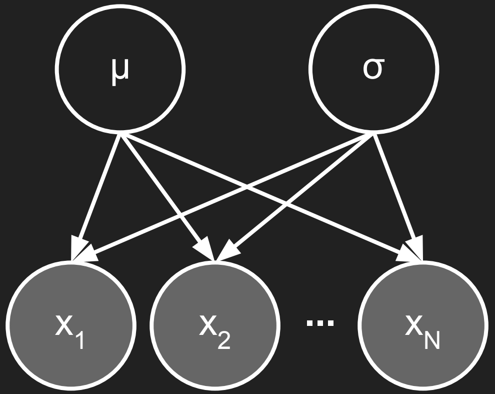


What is a Probabilistic Model?

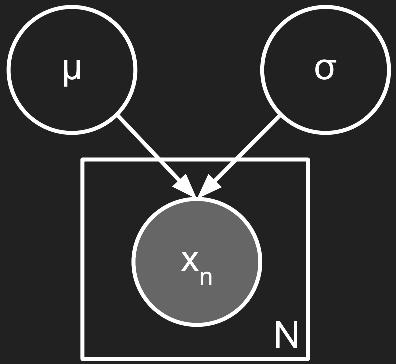
$\mu \sim \text{Normal}(0, 10)$

$\sigma \sim \text{LogNormal}(0, 5)$

$x_1,\ldots,x_N \overset{\text{iid}}{\sim} \text{Normal}(\mu, \sigma)$



these are the same

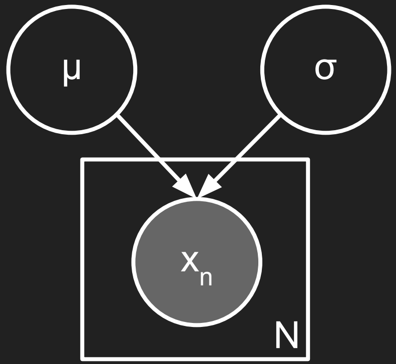


"plate" notation

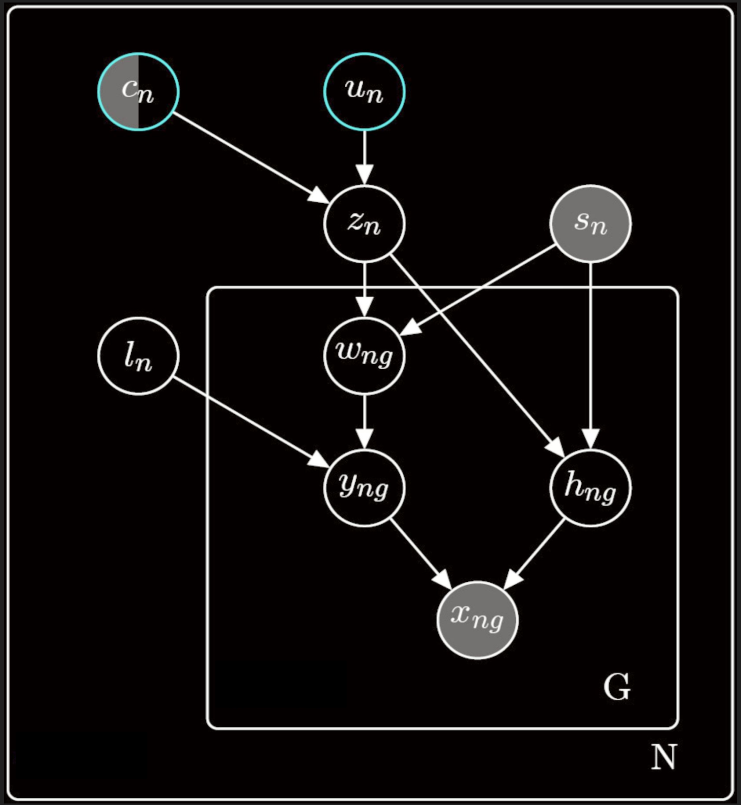
Why are Probabilistic Models useful?

Bayesian models are great at:

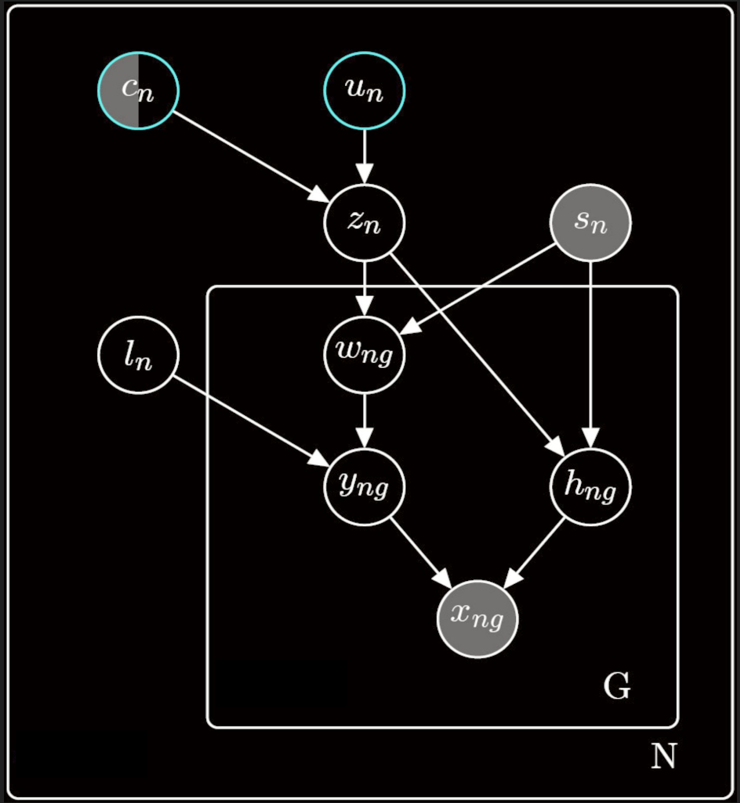
- incorporating noisy data
- fusing data
- handling uncertainty
- expressing prior knowledge



Express probabilistic models as programs

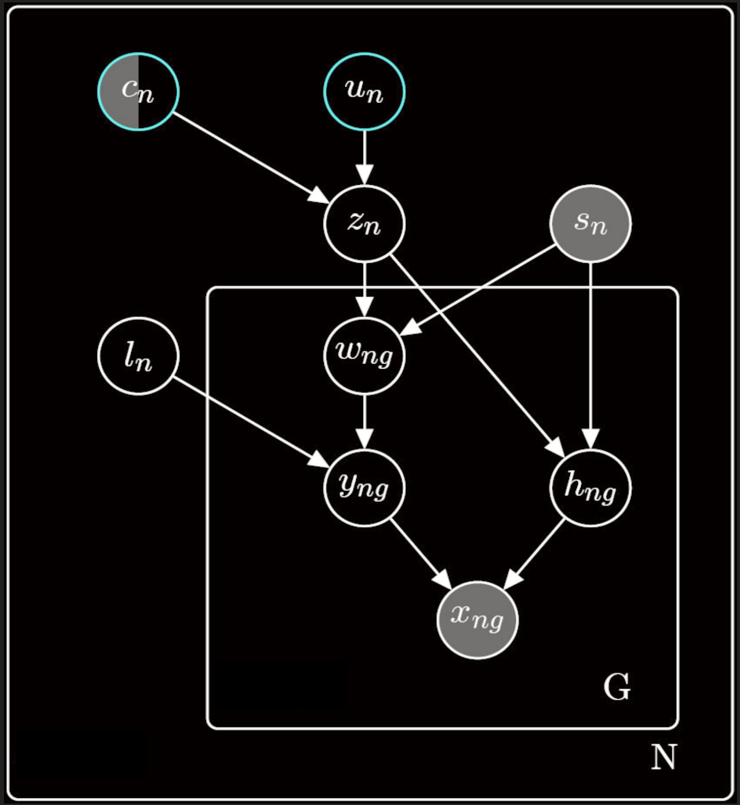


Express probabilistic models as programs



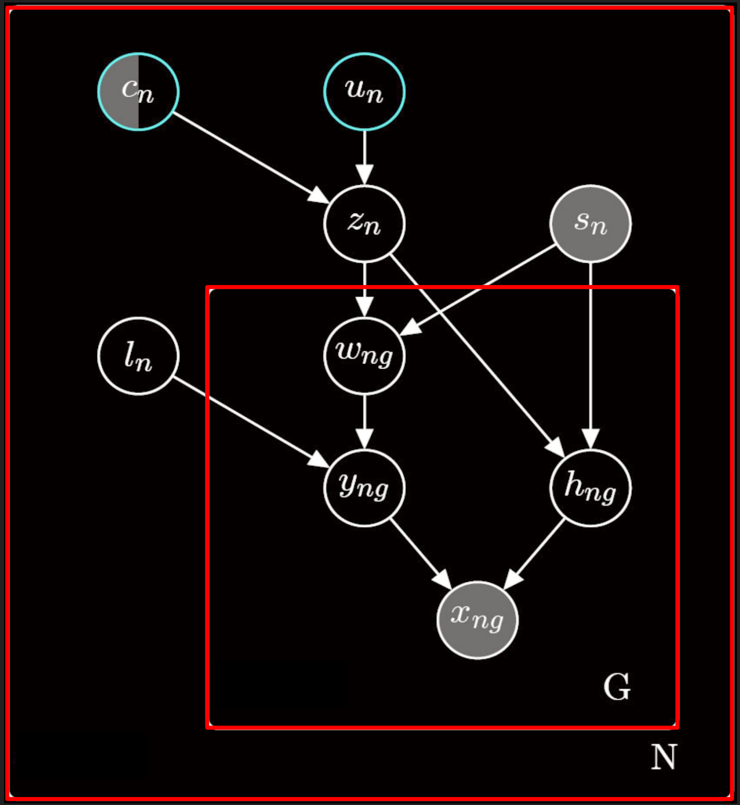

```
def model(s,x):
  for n in range(N):
    c = sample(c_dist())
    u = sample(u_dist())
    l = sample(l_dist())
    z = sample(z_dist(c,u))
    for g in range(G):
      w = sample(w_dist(z,s[n]))
      y = sample(y_dist(l,w))
      h = sample(h_dist(z,s[n]))
      sample(x_dist(y,h), obs=x[n,g])
```

Express probabilistic models as programs



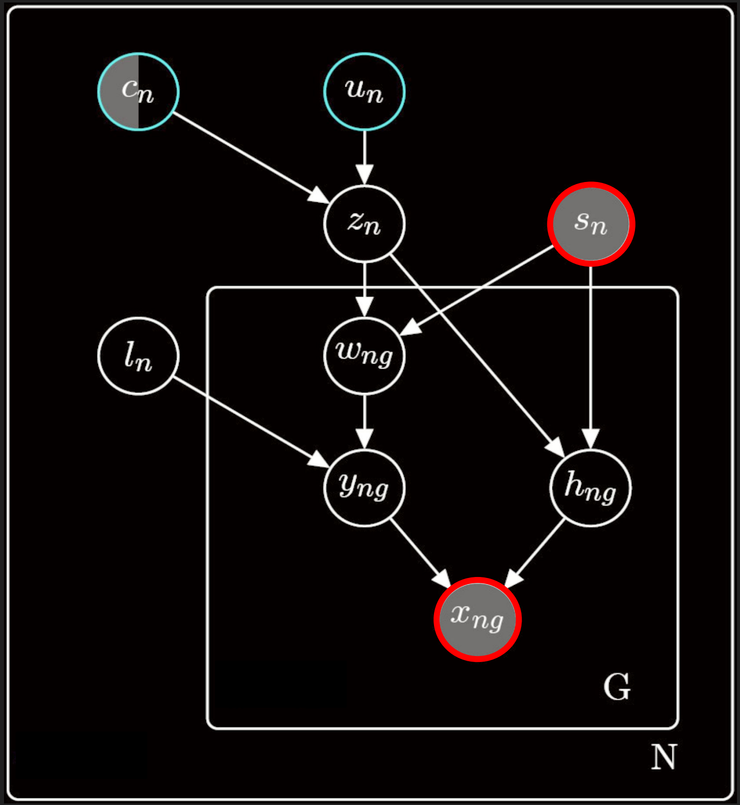

```
def model(s,x):
  for n in range(N):
    c = sample(c_dist())
    u = sample(u_dist())
    l = sample(l_dist())
    z = sample(z_dist(c,u))
    for g in range(G):
      w = sample(w_dist(z,s[n]))
      y = sample(y_dist(l,w))
      h = sample(h_dist(z,s[n]))
      sample(x_dist(y,h), obs=x[n,g])
```

Express probabilistic models as programs




```
def model(s,x):
  for n in range(N):
    c = sample(c_dist())
    u = sample(u_dist())
    l = sample(l_dist())
    z = sample(z_dist(c,u))
    for g in range(G):
      w = sample(w_dist(z,s[n]))
      y = sample(y_dist(l,w))
      h = sample(h_dist(z,s[n]))
      sample(x_dist(y,h), obs=x[n,g])
```

Express probabilistic models as programs

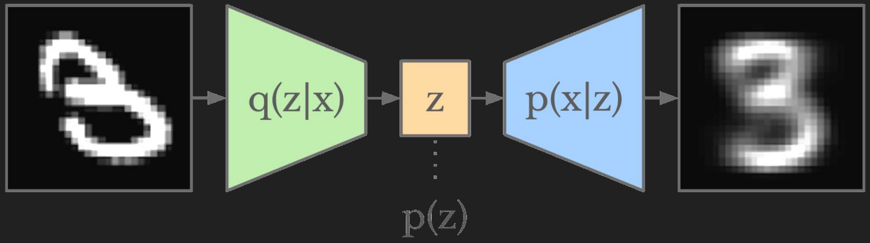



```
def model(s,x):
  for n in range(N):
    c = sample(c_dist())
    u = sample(u_dist())
    l = sample(l_dist())
    z = sample(z_dist(c,u))
    for g in range(G):
      w = sample(w_dist(z,s[n]))
      y = sample(y_dist(l,w))
      h = sample(h_dist(z,s[n]))
      sample(x_dist(y,h), obs=x[n,g])
```

Inference via Automatic Differentiation

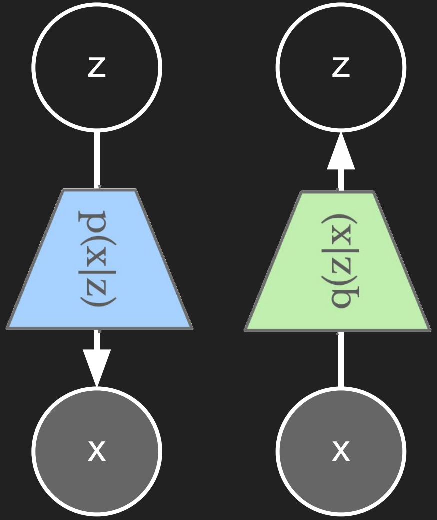
- Hamiltonian Monte Carlo (Neal 1996)
- No U-Turn Sampler (Hoffman & Gelman 2011)

- Automatic Differentiation VI (Ranganath et al. 2014, Kucukelbir et al. 2016)
- Stochastic Variational Inference (Hoffman et al. 2013)
- Variational autoencoders (VAEs) (Kingma & Welling 2014)

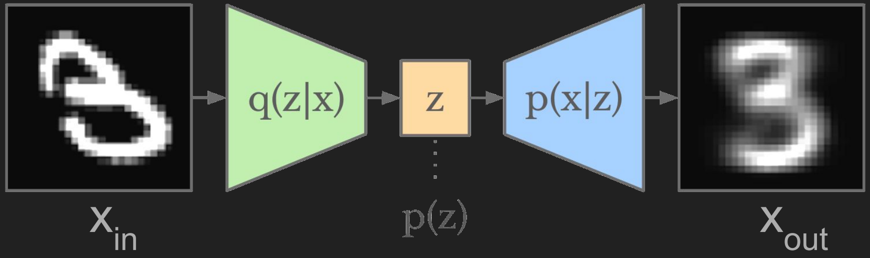


(Danijar Hafner)

VAEs are pairs of probabilistic models



"a generative model +
an inference model"



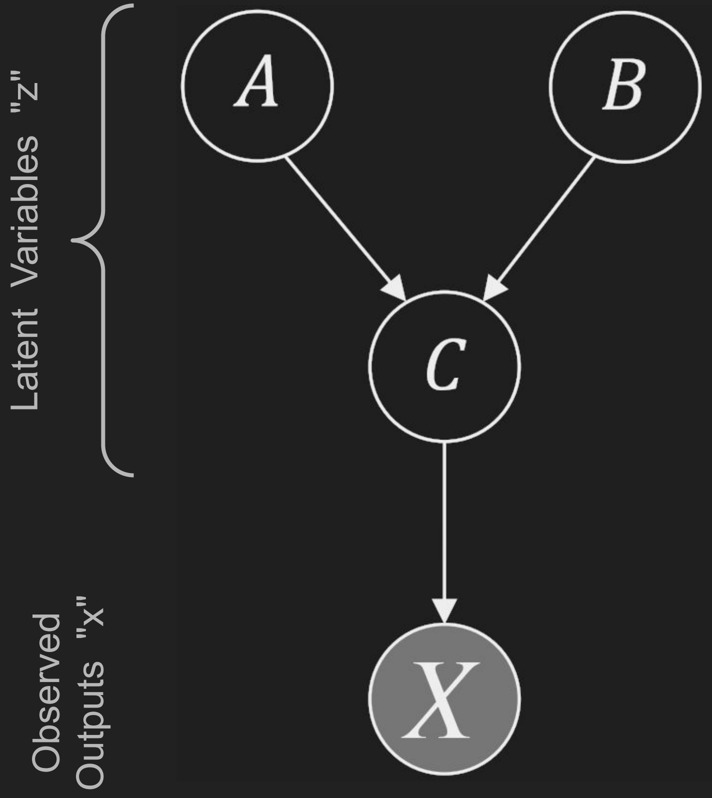
"bottleneck network
with stochastic layers"

Generative models are simulators

latents ⟶ outputs

decoder distribution p(x|z)

```
A ~ Normal(0, 1)
B ~ LogNormal(0, 1)
C ~ LogNormal(A, B)
X ~ Poisson(C)
```



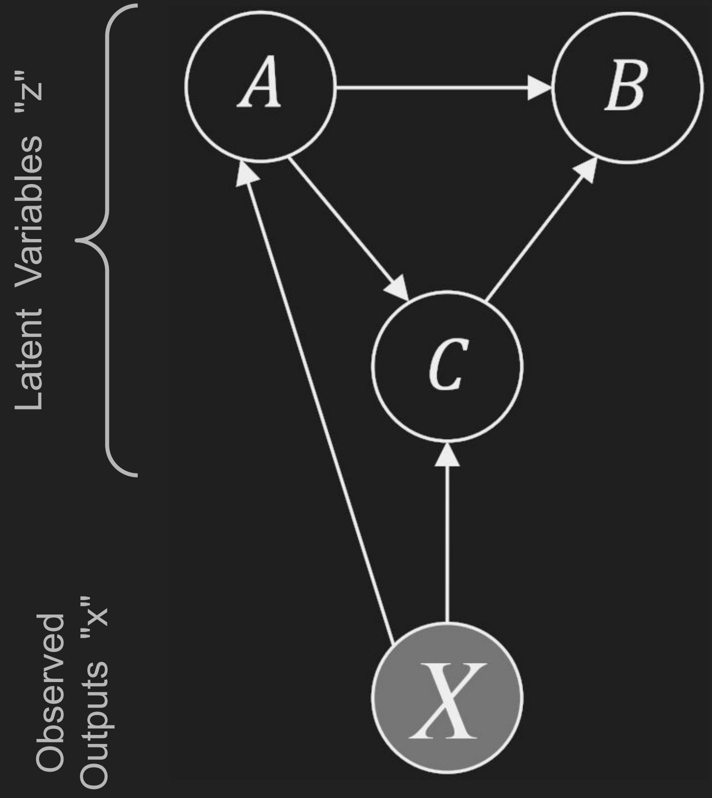
Sampling is easy!

Inference algorithms invert simulators

outputs → latents

encoder distribution q(z; x)

```
A ~ ComplexDistribution1(X)
C ~ ComplexDistribution2(X, A)
B ~ ComplexDistribution3(A, C)
```

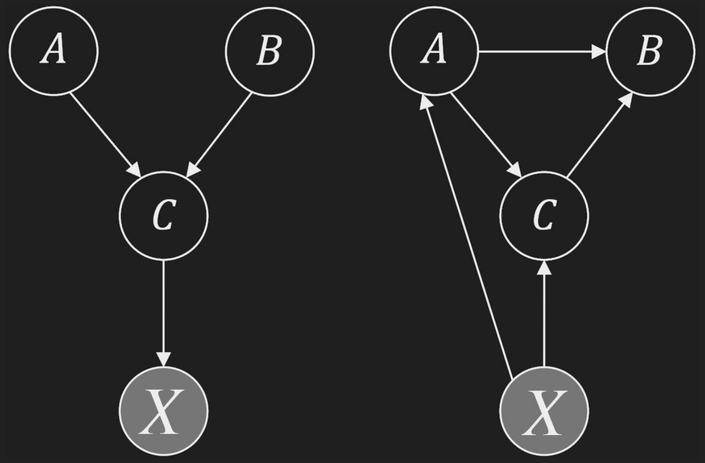


Inference is hard!

VAEs suggest a general inference recipe:

1. Write a generative model, "p"
   latents → outputs
2. Write an inference model, "q"
   outputs → latents

3. Train on output data "x"
4. Predict latents from inverse model
   z ~ q(z;x)
5. Minimize -ELBO = KL(q(z;x) || p(z|x)) + C
6. Compute gradients with AD
7. Update parameters with SGD



generative model
p(a,b,c,x)

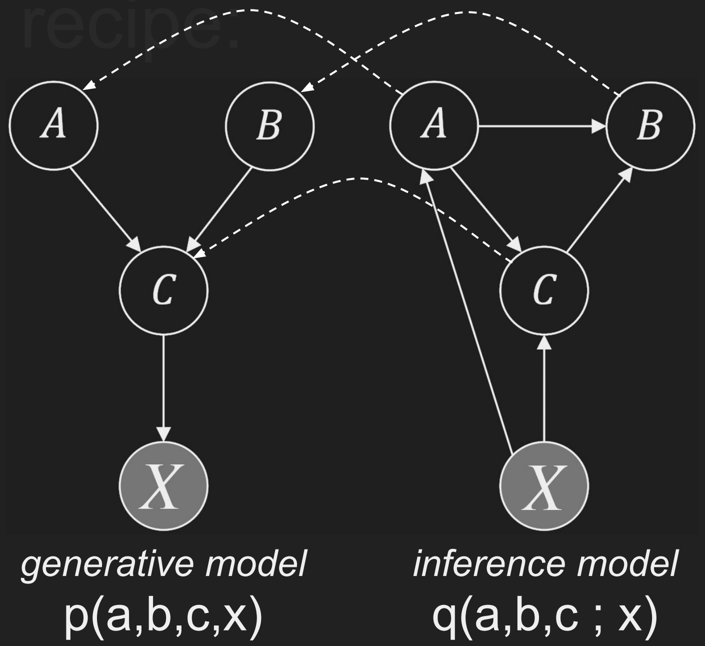
inference model
q(a,b,c ; x)

(Webb et al. 2017)

VAEs suggest a general inference recipe:

1. Write a generative model, "p"
   latents → outputs
2. Write an inference model, "q"
   outputs → latents

3. Train on output data "x"
4. Predict latents from inverse model
   z ~ q(z;x)
5. Minimize -ELBO = KL(q(z;x) || p(z|x)) + C
6. Compute gradients with AD
7. Update parameters with SGD

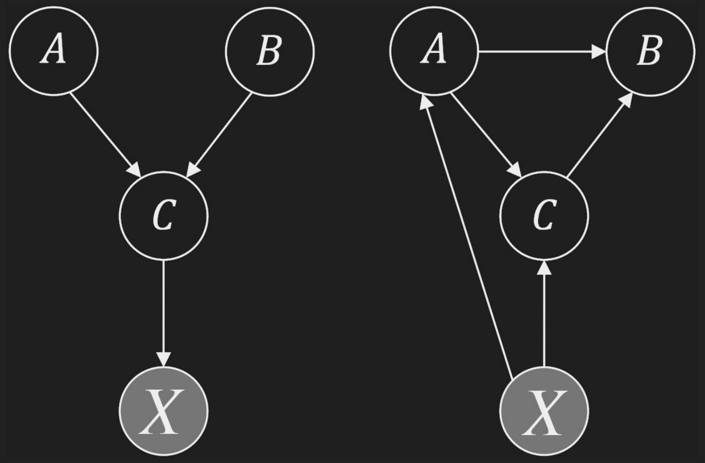


(Webb et al. 2017)

VAEs suggest a general inference recipe:

1. Write a generative model, "p"
   latents → outputs
2. Write an inference model, "q"
   outputs → latents

3. Train on output data "x"
4. Predict latents from inverse model
   z ~ q(z;x)
5. Minimize -ELBO = KL(q(z;x) || p(z|x)) + C
6. Compute gradients with AD
7. Update parameters with SGD



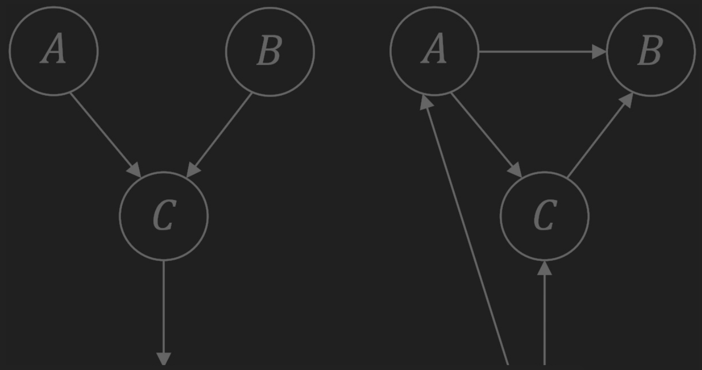
generative model
p(a,b,c,x)

inference model
q(a,b,c ; x)

(Webb et al. 2017)

VAEs suggest a general inference recipe:

1. Write a generative model, "p"
   latents → outputs
2. Write an inference model, "q"
   outputs → latents



3. Train on output data "x"
4. Predict latents from inverse model
   z ~ q(z;x)
5. Minimize -ELBO = KL(q(z;x) || p(z|x)) + C
6. Compute gradients with AD
7. Update parameters with SGD

This is independent of models (p,q).

Let's implement it once in a library.

(Webb et al. 2017)

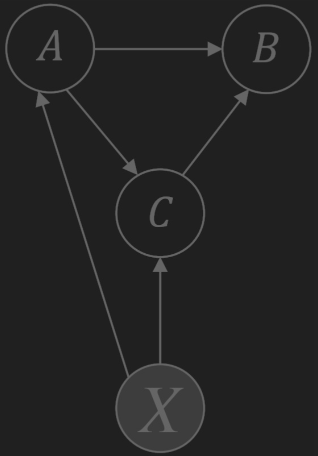
VAEs suggest a general inference recipe:

1. Write a generative model, "p"
   latents → outputs
2. Write an inference model, "q"
   outputs → latents

This is the only problem-specific part.

Let's make it extremely flexible.

3. Train on output data "x"
4. Predict latents from inverse model
   z ~ q(z;x)
5. Minimize -ELBO = KL(q(z;x) || p(z|x)) + C
6. Compute gradients with AD
7. Update parameters with SGD



generative model
p(a,b,c,x)

inference model
q(a,b,c ; x)

(Webb et al. 2017)

Pyro's interface

Modeling Language

Inference Algorithms

Pyro's interface

Modeling Language

`pyro.sample`
`pyro.param`
`pyro.plate`
`pyro.factor`
`pyro.deterministic`
`...`

Inference Algorithms

`SVI` (variational inference)
`ELBO`
`NUTS`
`HMC`
`SMC`
`...`

Pyro extends Python with primitives

```
x = pyro.sample("x", Bernoulli(0.5))
assert isinstance(x, torch.Tensor)

pyro.sample("y", Normal(x, 1.),
            obs=y)

theta = pyro.param("theta", torch.ones(100),
                   constraint=positive)
```

Pyro extends Python with primitives

```
for i in pyro.plate("data", len(data), batch_size):
    pyro.sample(f"data_{i}", fun(x), obs=data[i])


with pyro.plate("data", len(data), batch_size) as i:
    pyro.sample("data", fun(x), obs=data[i])
```

Pyro models are Python functions

```python
def model(data):
    p = pyro.param("p", torch.ones(10)/10, constraint=simplex)
    c = pyro.sample("c", Categorical(p))
    if c > 0:
        pyro.sample("obs", Normal(helper(c - 1), 1.),
                    obs=data)

def helper(c):
    x = pyro.sample("x" Normal(0., 10.))
    return x[c]
```

Pyro inference looks like neural net training

```python
# Fit a model.
guide = AutoNormal(model)  # or a custom guide
optim = Adam({"lr": 1e-3})
svi = SVI(model, guide, optim, Trace_ELBO())
for step in range(1000):
    svi.step(data)


# Draw samples from the posterior.
samples = Predictive(model, guide=guide)(data)
```

Deep Probabilistic Programming

- Modeling in Python (Tran et al. 2017, Bingham et al. 2018)
- Built on well-engineered libraries for arrays & neural nets:
  Pyro/PyTorch, NumPyro/JAX, Edward/Tensorflow, ...
- Recent tricks:
  neural baselines (Mnih & Gregor 2014)
  normalizing Flows (Rezende & Mohamad 2015)
  sticking the landing (Roeder et al. 2017)
  delayed sampling (Murray et al. 2017)
  variational OED (Foster et al. 2018)
  tensor monte carlo (Aitchison 2018)
  DiCE estimator (Foerster et al. 2018)
  tensor variable elimination (Obermeyer et al. 2019)
  parallel-scan filtering (Särkkä & García-Fernández 2019)
  reparameterisation effects (Gorinova et al. 2019)
  functional tensors (Obermeyer et al. 2019)
  Stein VI (Al-Sibahi & Rønning 2020)

What we're working on

New MCMC algorithms
- discrete latent variables
- data subsampling

Automating variational inference
- structured variational distributions
- automatic reparametrization

Normalizing flows

Thank you! https://pyro.ai


Du Phan


Eli Bingham


Martin Jankowiak


Neeraj Pradhan


J.P. Chen


Stefan Webb


Noah Goodman


Fritz Obermeyer