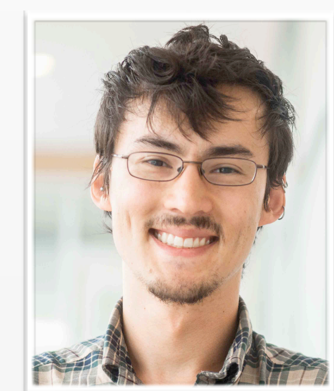
Compositional Methods for Learning and Inference in Deep Probabilistic Programs

Jan-Willem van de Meent

Hao Wu

Heiko Zimmermann

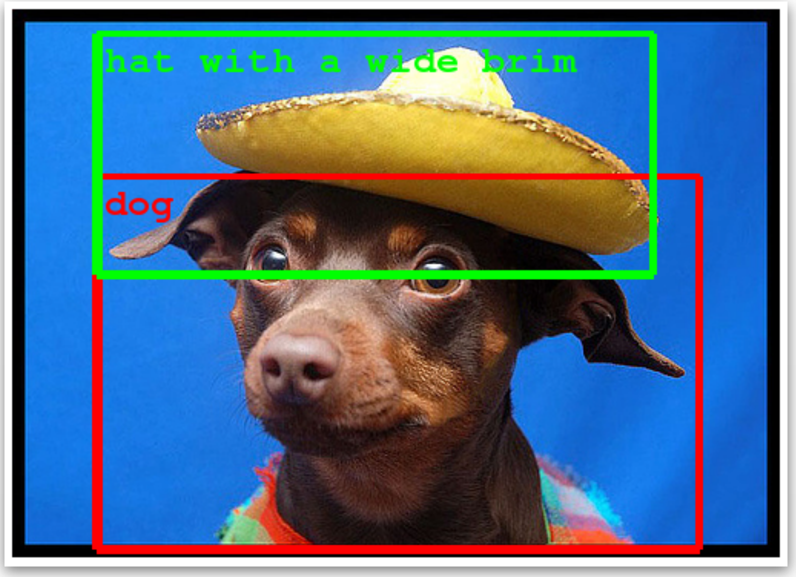
Eli Sennesh



Sam Stites

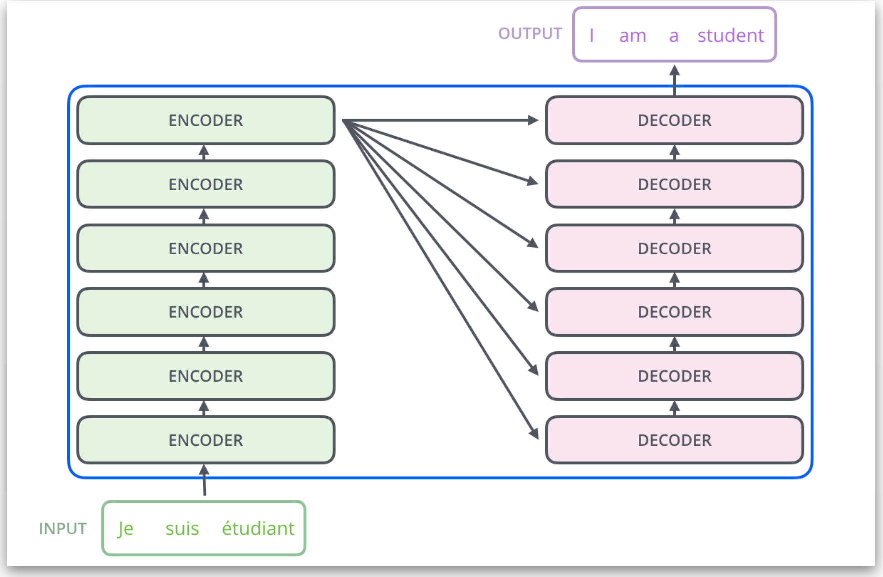
Deep Learning Success Stories

Computer Vision



14M images (ImageNet)
Annotations available

Natural Language



Very large corpora of text
(can self-supervise)

Reinforcement Learning

4.9M games (Self-play)
Clear definition of success

Ingredients for success

1. Abundance of (labeled) data and compute
2. A well-defined general notion of utility

Do we still need models?

The Bitter Lesson

Rich Sutton, March 13, 2019

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin.

Do we still need models or just more data and compute?

Max Welling, April 20, 2019

When you need to generalize to new domains, i.e. extrapolate away from the data, you will need a generative model

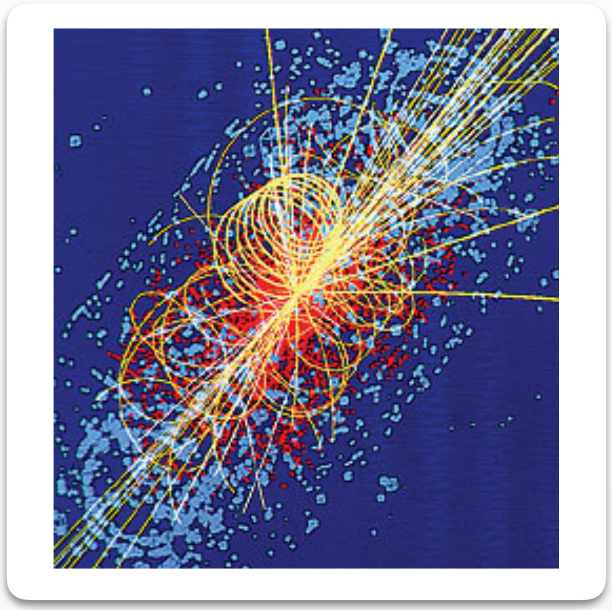
http://www.incompleteideas.net/IncIdeas/BitterLesson.html

https://staff.fnwi.uva.nl/m.welling/wp-content/uploads/Model-versus-Data-AI-1.pdf

Do we ~~still need~~ models?

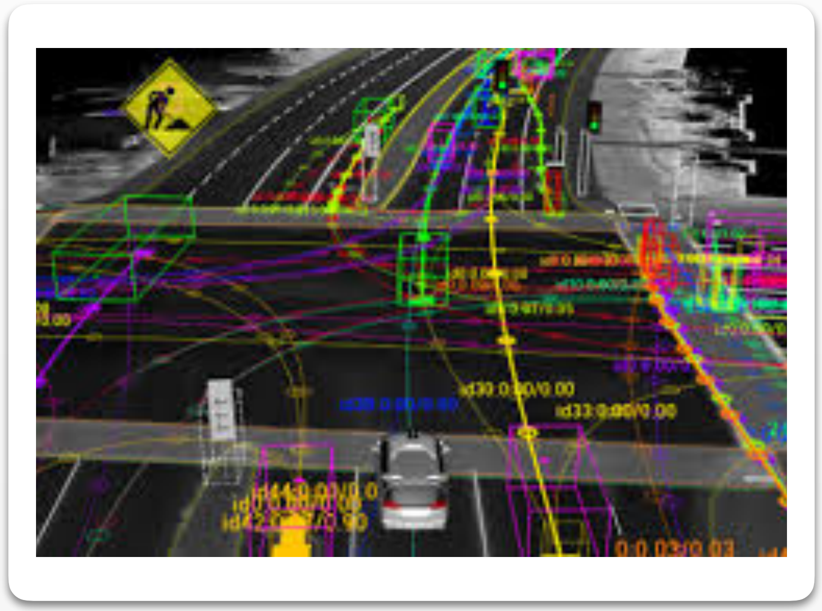
When are models useful?

Science & Engineering



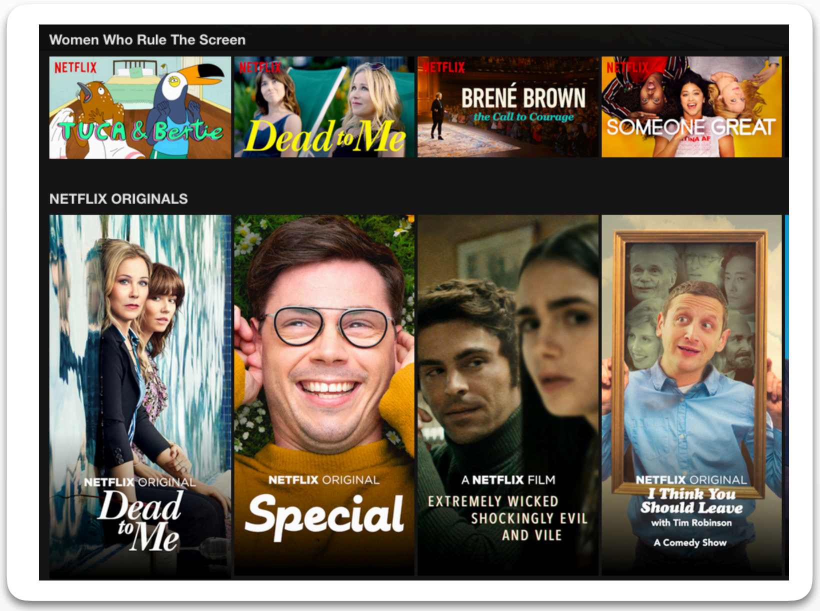
High quality models and/or limited data

Autonomous Vehicles



Generalization to long tail events

Recommendation



Large collection of small-data problems

We need *inductive biases* that

1. Improve generalization
2. Safe-guard against overconfident predictions

Deep Probabilistic Models

Deep Learning

- High-capacity models
- Scalable to large datasets
- Easy to try new models

Probabilistic Programming

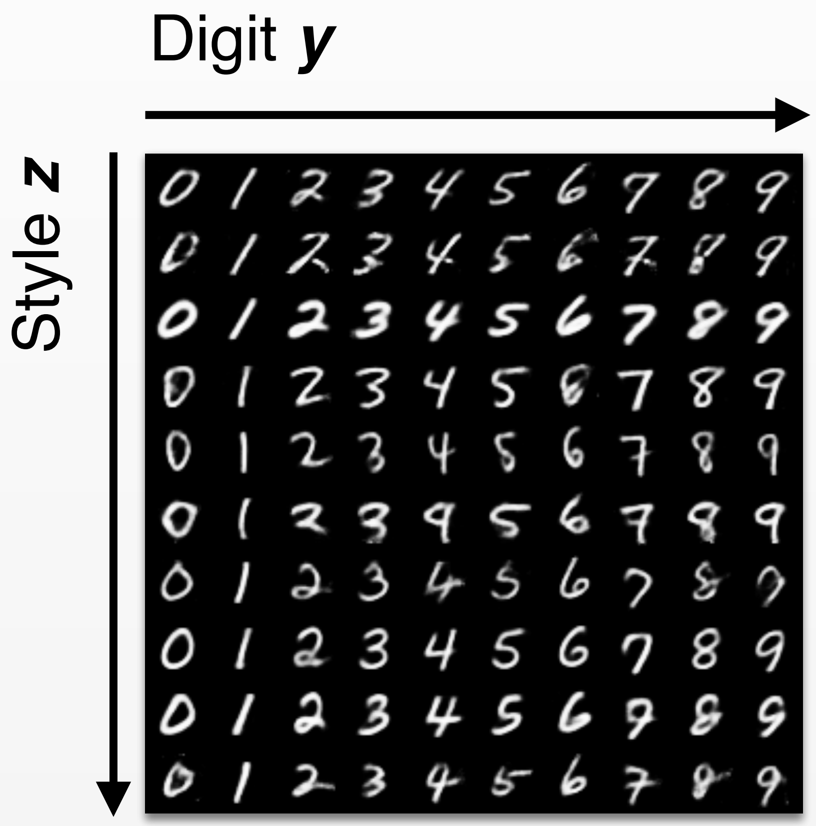
- Programs as inductive biases
- Structured, interpretable
- Also easy to try new models

SGD + AutoDiff
(very general)

Monte Carlo Methods
(more model specific)

Stochastic Variational Inference
(learn proposals using neural networks)

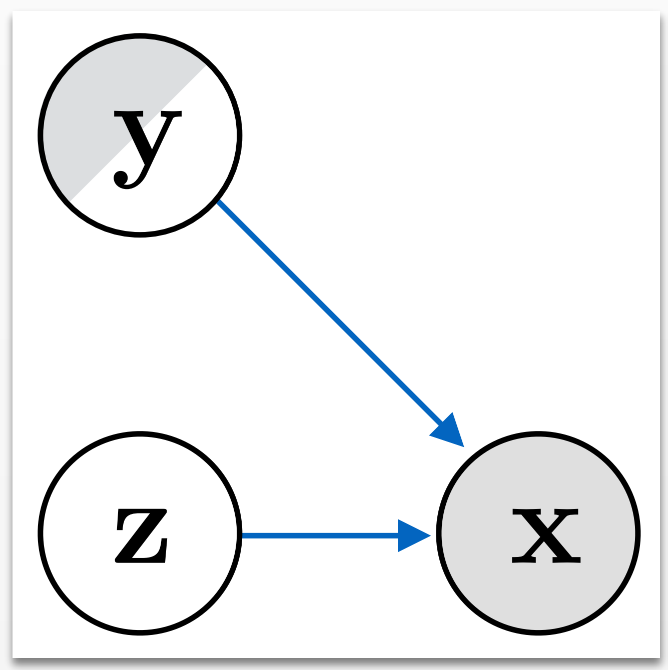
Structured Variational Autoencoders



Goal: learn “disentangled” representation for ***y*** and ***z***

Generative Model (Decoder)

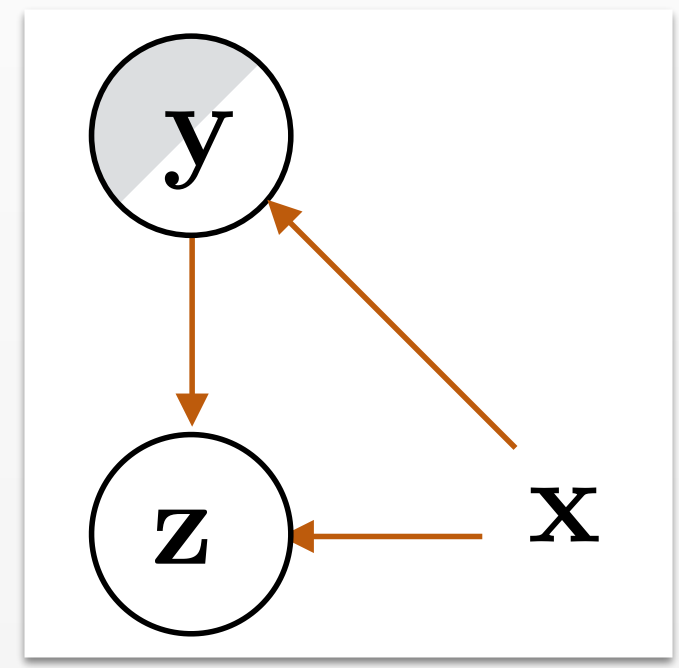
$p_\theta(\boldsymbol{x} \mid \boldsymbol{y}, \boldsymbol{z})\, p(\boldsymbol{y}) p(\boldsymbol{z})$



Assume independence between digit **y** and style **z**

Inference Model (Encoder)

$q_\phi(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})\, q(\boldsymbol{x})$



Infer **y** from pixels **x**, and **z** from **y** and **x**

[Kingma, Mohamed, Jimenez-Rezende, Welling, NIPS 2014]

Deep Probabilistic Programs

Generative Model (Decoder)

```
class Decoder(torch.nn.Module):
  def __init__(self, x_sz, h_sz, y_sz, z_sz):
    # intializes layers: h, x_mean, ...
    ...

  def forward(self, x, q):
    p = probtorch.Trace()
    y = p.concrete(self.y_log_weights, 0.66,
                   value=q['y'], name='y')
    z = p.normal(0.0, 1.0,
                 value=q['z'], name='z')
    h = self.h(torch.cat([y, z], -1))
    x = p.loss(self.bce,
               self.x_mean(h), x,
               name='x')
    return p
```

Inference Model (Encoder)

```
class Encoder(torch.nn.Module):
  def __init__(self, x_sz, h_sz, y_sz, z_sz):
    # intializes layers: h, y_log_weights, ...
    ...

  def forward(self, x, y_values=None):
    q = probtorch.Trace()
    h = self.h(x)
    y = q.concrete(
          self.y_log_weights(h), 0.66,
          value=y_values, name='y')
    hy = torch.cat([h, y], -1)
    z = q.normal(self.z_mean(hy),
                 self.z_std(hy),
                 name='z')
    return q
```

Edward

Probabilistic Torch

Pyro

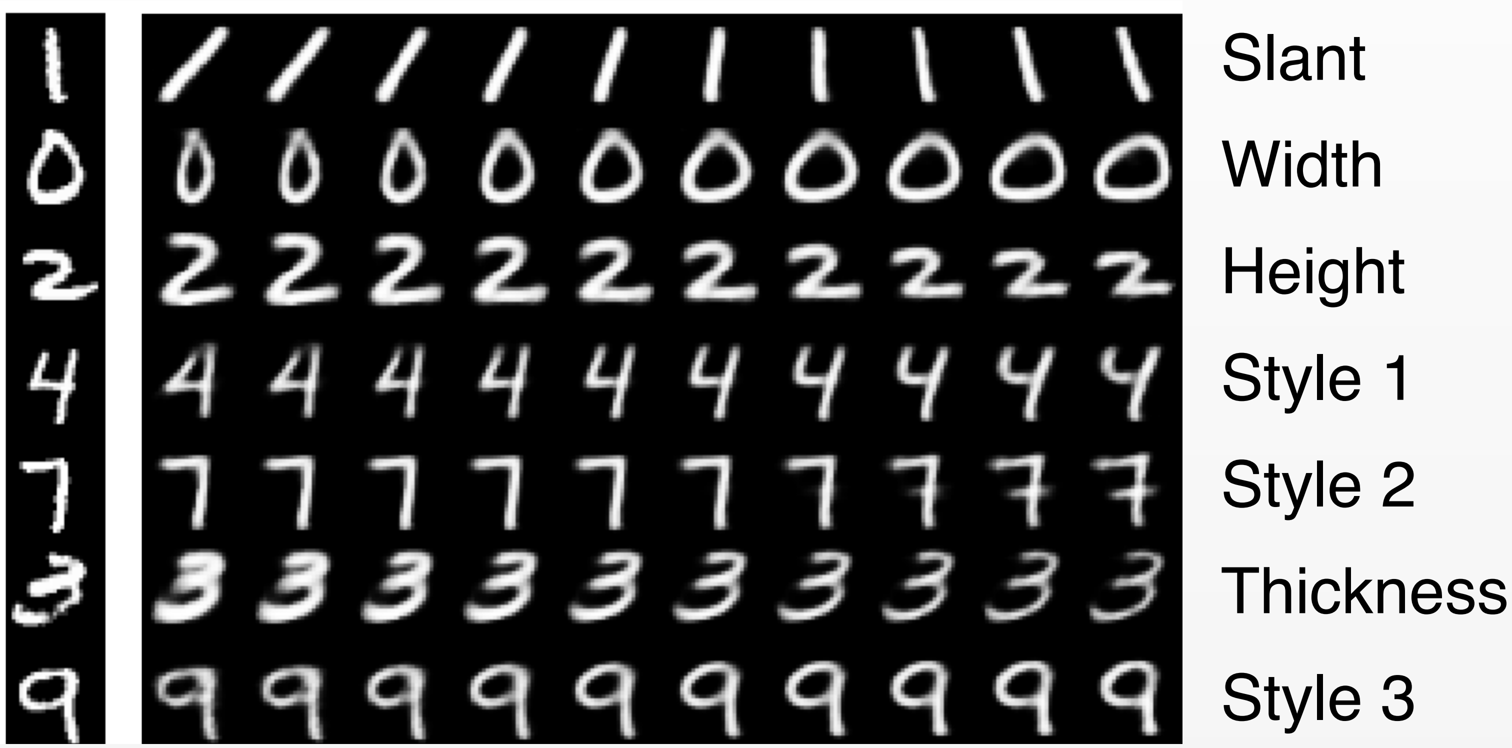
https://github.com/bleilab/edward

https://github.com/probtorch/probtorch

https://github.com/uber/pyro

Learned Representations (Unsupervised)

Style Variables

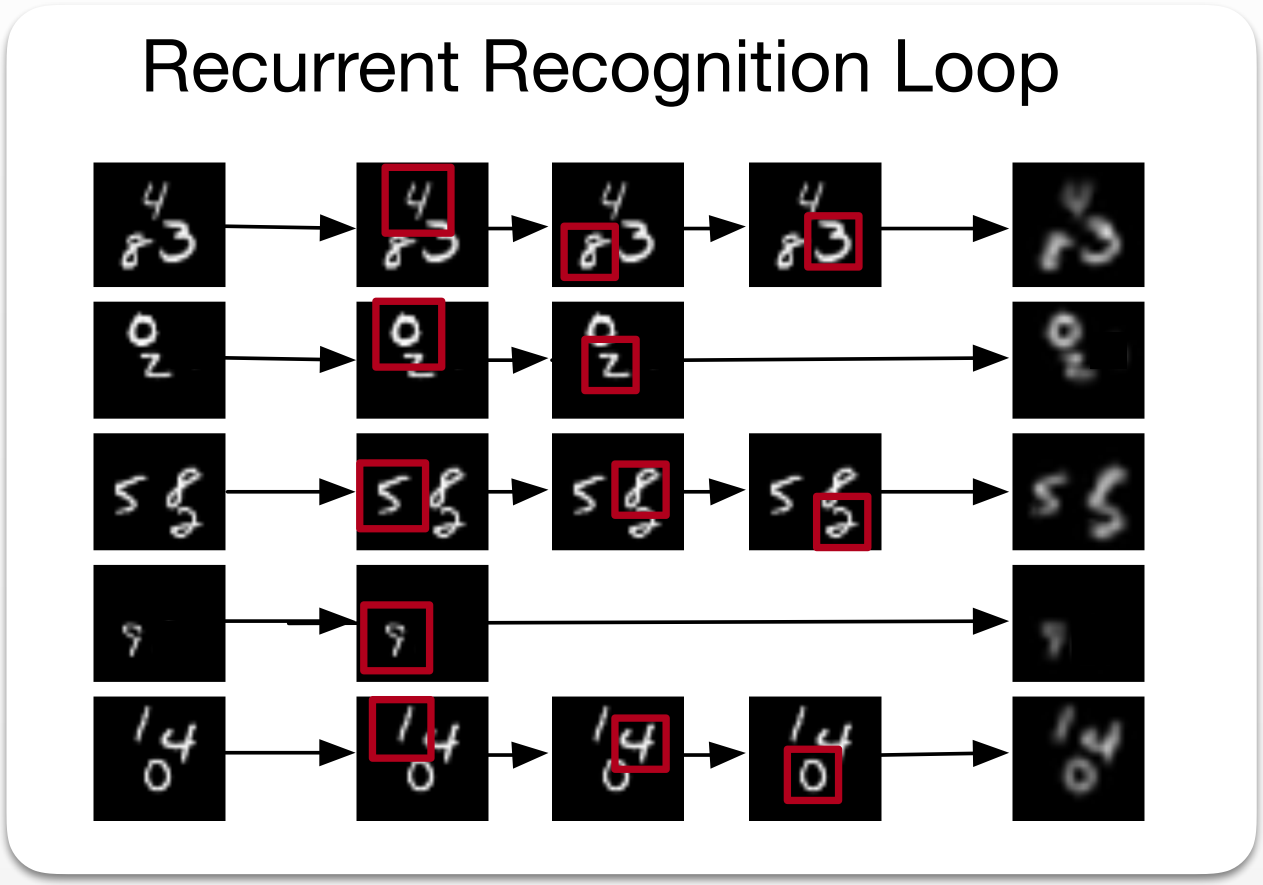


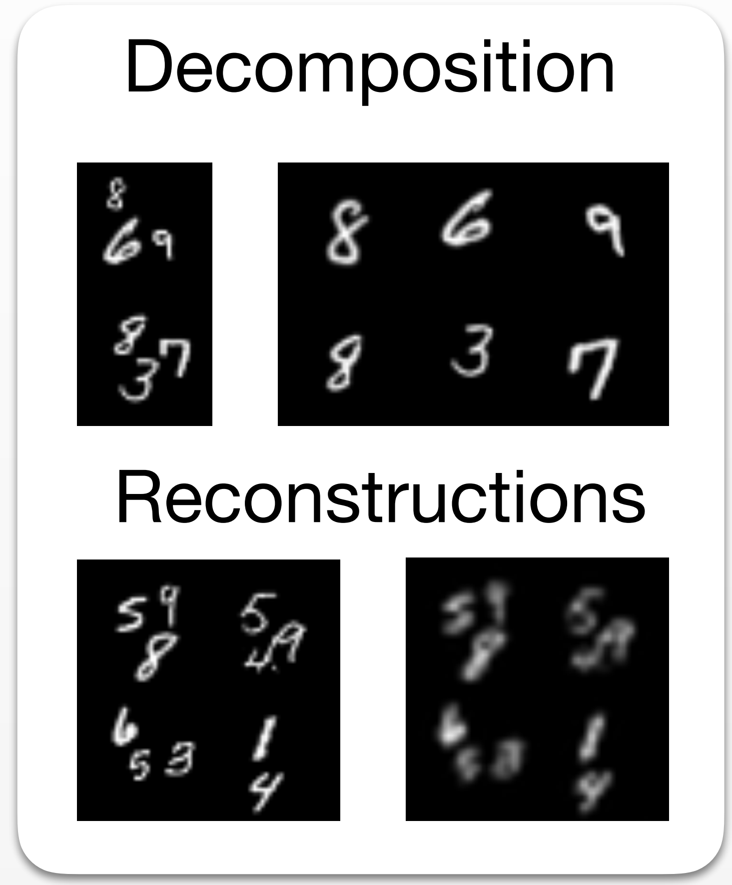
Generalization

Inductive Bias: Style features are uncorrelated with digit label, as well as with other features.

[Esmaeli, Wu, Jain, Bozkurt, Siddharth, Paige, Brooks, Dy, van de Meent, AISTATS 2019]

Model Composition

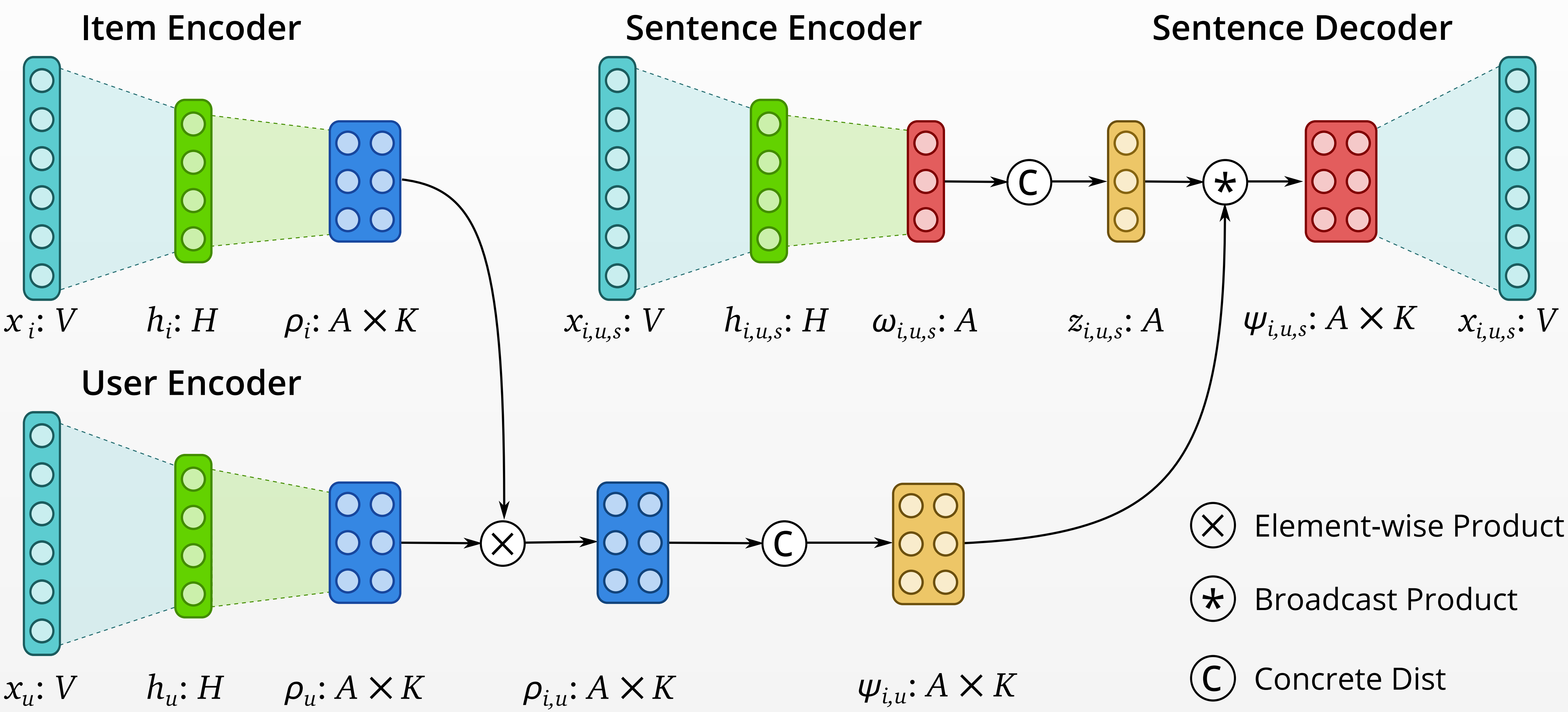




Idea: Embed model for individual MNIST digits in a recurrent model for multiple object detection

[Siddharth*, Paige*, van de Meent*, Desmaison, Wood, Goodman, Kohli, Torr, NIPS 2017]

Example: Modeling Aspects in Reviews



Learn aspect-based representations of
users, items, and reviews (*fully unsupervised*)

[Esmaeli, Huang, Wallace, van de Meent, *AISTATS 2019*]

Example: Modeling Aspects in Reviews

Data: Beer reviews

Amber brown in color with very little head but a nice ring. Nicely carbonated. Smells like a camp fire, malts have a good sweet character with an abundance of smoke. Taste is quite good with smokiness being pungent but not overwelming. A sweet tasting bock with smokiness coming through around mid drink with a smooth mellow finish. A good warming smoky beer.

Aspects: Look, Mouthfeel, Aroma, Taste, Overall

[Esmaeli, Huang, Wallace, van de Meent, *AISTATS 2019*]

Example: Modeling Aspects in Reviews

Appearance		Aroma-Taste		Palate		Semantic	
golden	black	roasted	citrus	mouthfeel	mouthfeel	lagers	try
yellow	tans	coffee	grapefruit	bodied	watery	heineken	hype
white	glass	vanilla	pine	smooth	rjt	macro	recommend
orange	pour	chocolate	hops	carbonation	bodied	import	overall
hazy	head	bourbon	lemon	medium	refreshing	euro	founders
color	pitch	oak	floral	drinkability	carbonation	lager	favorite
gold	lacing	malts	clove	drinkable	crisp	bmc	stouts
copper	color	sweet	malt	alcohol	dry	worse	stout
straw	brown	aroma	aroma	finish	finish	bad	ipa
amber	ginger	malt	grass	mouth	thirst	skunky	cheers

Model learns topics that group into aspects
(*fully unsupervised*)

[Esmaeli, Huang, Wallace, van de Meent, *AISTATS 2019*]

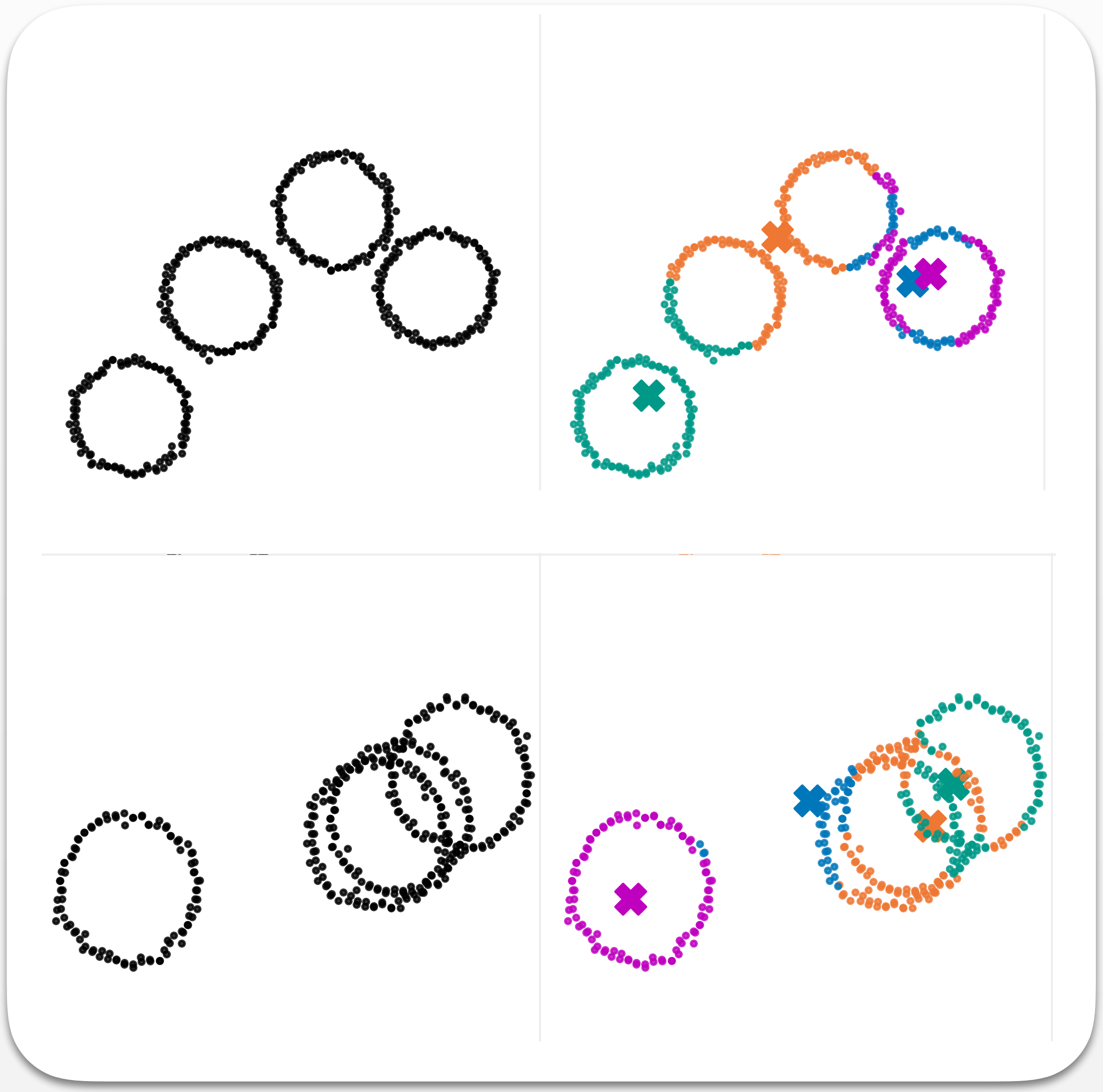
Where are we today?

Structured VAEs work well for O(10) variables

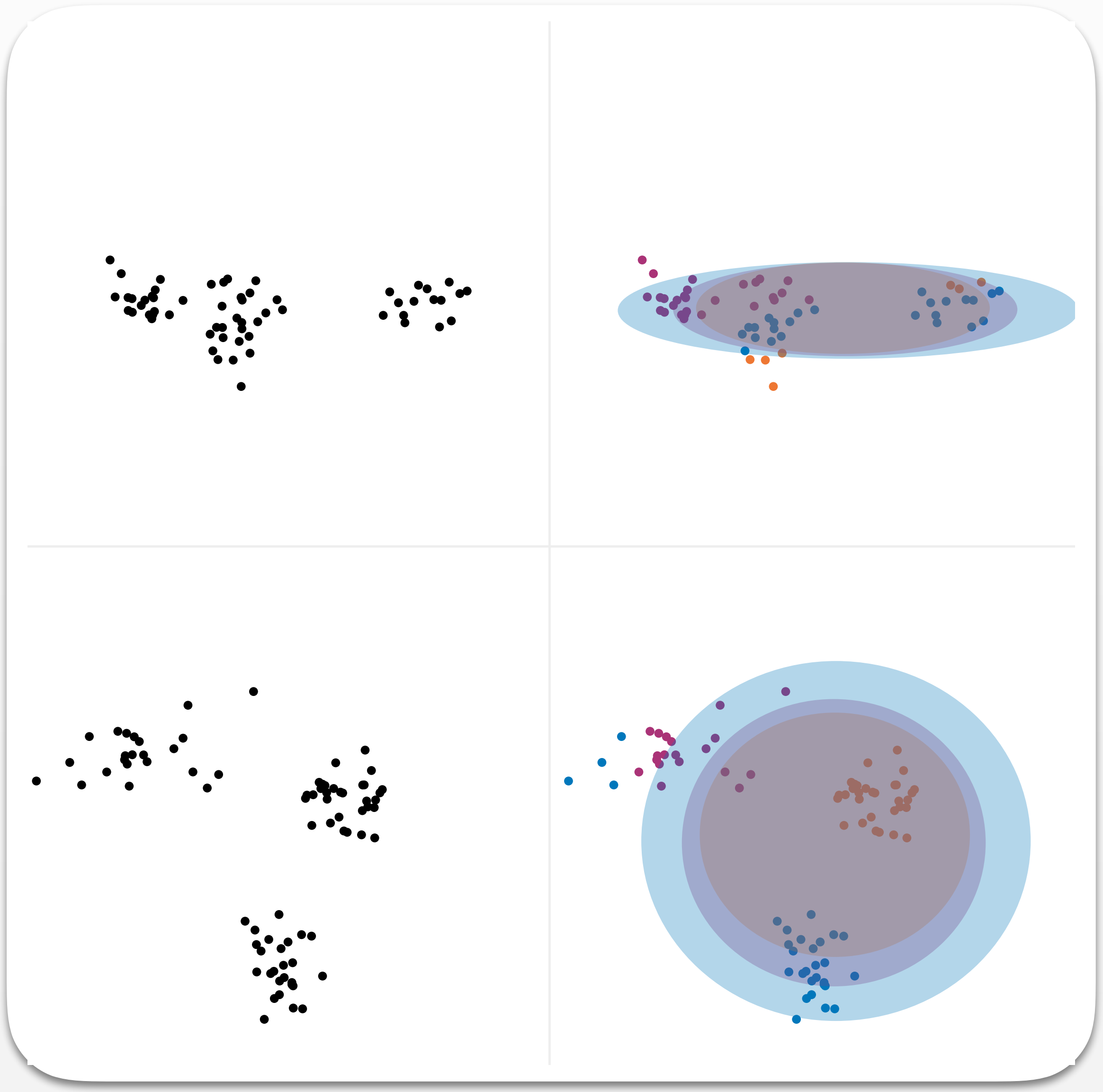
- Object recognition (~3 objects)
  [Eslami et al, NeurIPS 2016]
- Object tracking (~3 objects, ~10 frames)
  [Kosiorek et al, NeurIPS 2018]
- Product Reviews (~10 sentences, ~10 aspects)
  [Esmaeli et al, AISTATS 2019]

Still Hard: Clustering

Data Inference



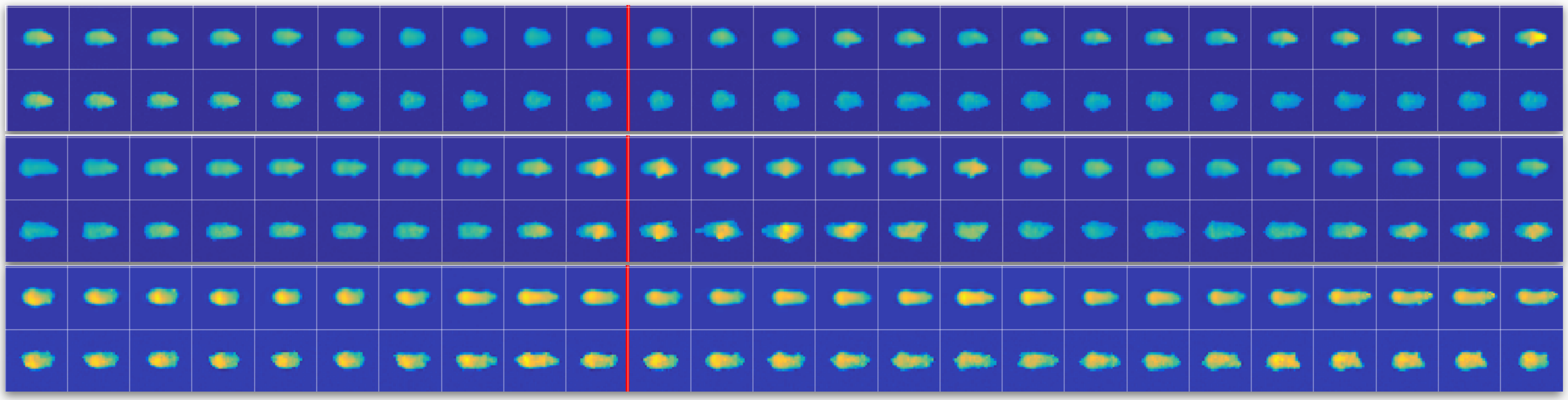
Data Inference



We need methods that scale to > O(100) variables

Paths towards Scaling up Inference

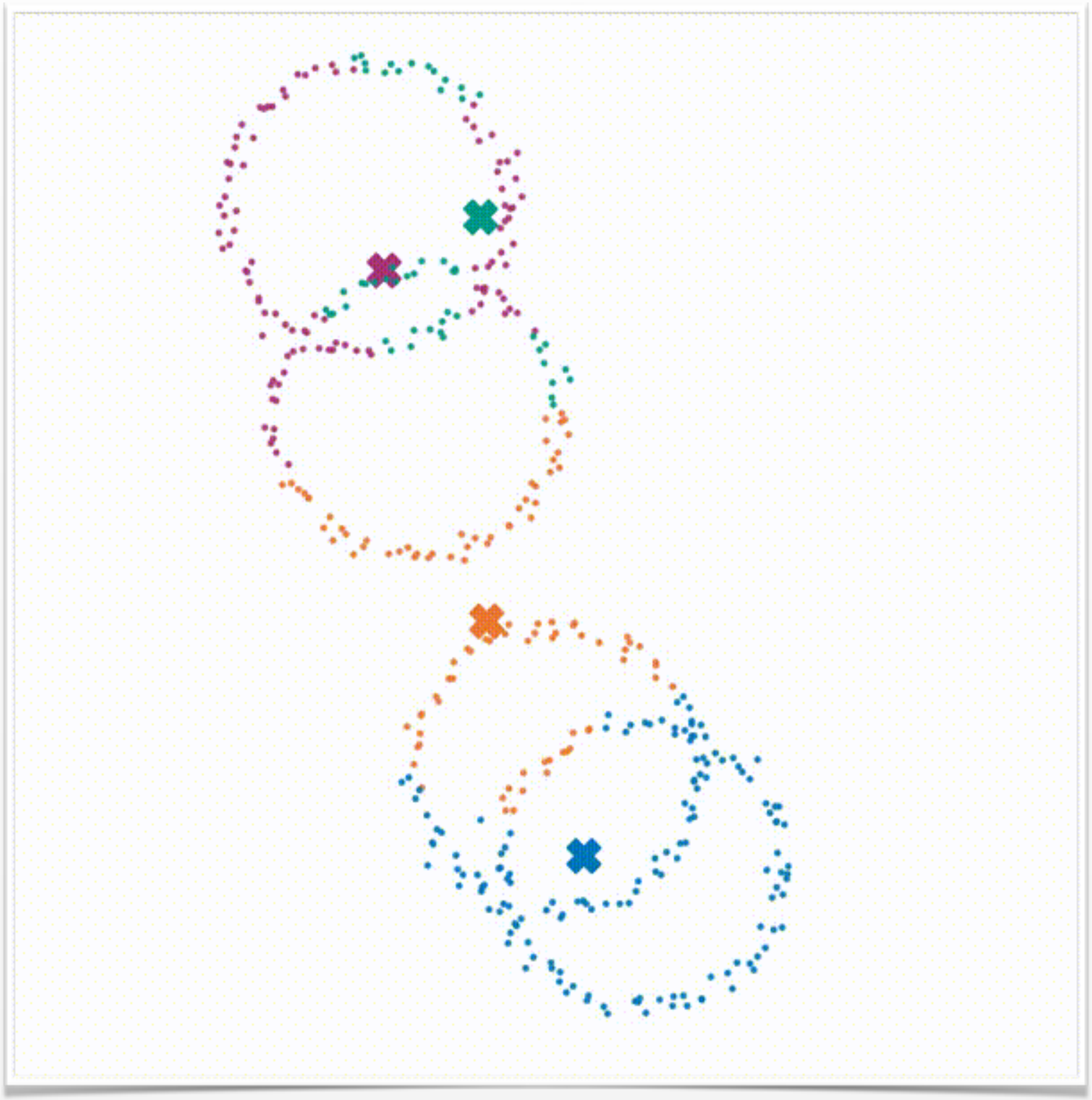
Example: Deep State Space Models for Activity Recognition in Mice

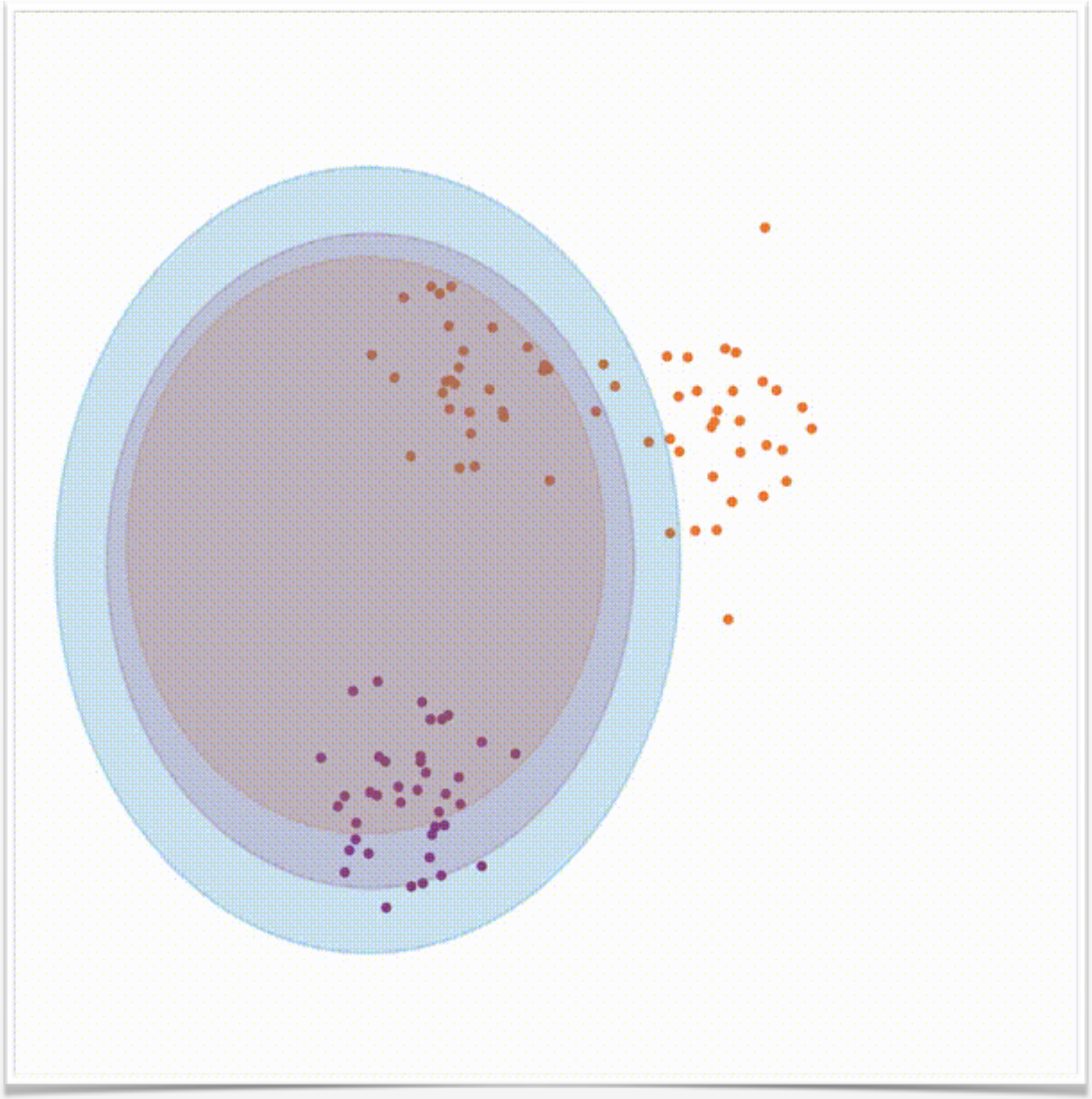


1. **Generative Model:** Exponential family + Neural likelihood
2. **Inference:** Variational EM + Gradient descent

[Johnson, Duvenaud, Wiltschko, Adams, Datta, NIPS 2016] [Wiltschko, Neuron 2015]

Amortized Gibbs Samplers





Intuition: Same idea as Gibbs sampling or Variational EM, but learn proposals for conditional updates

[Wu, Zimmermann, Sennesh, Le, van de Meent, arXiv 2019]

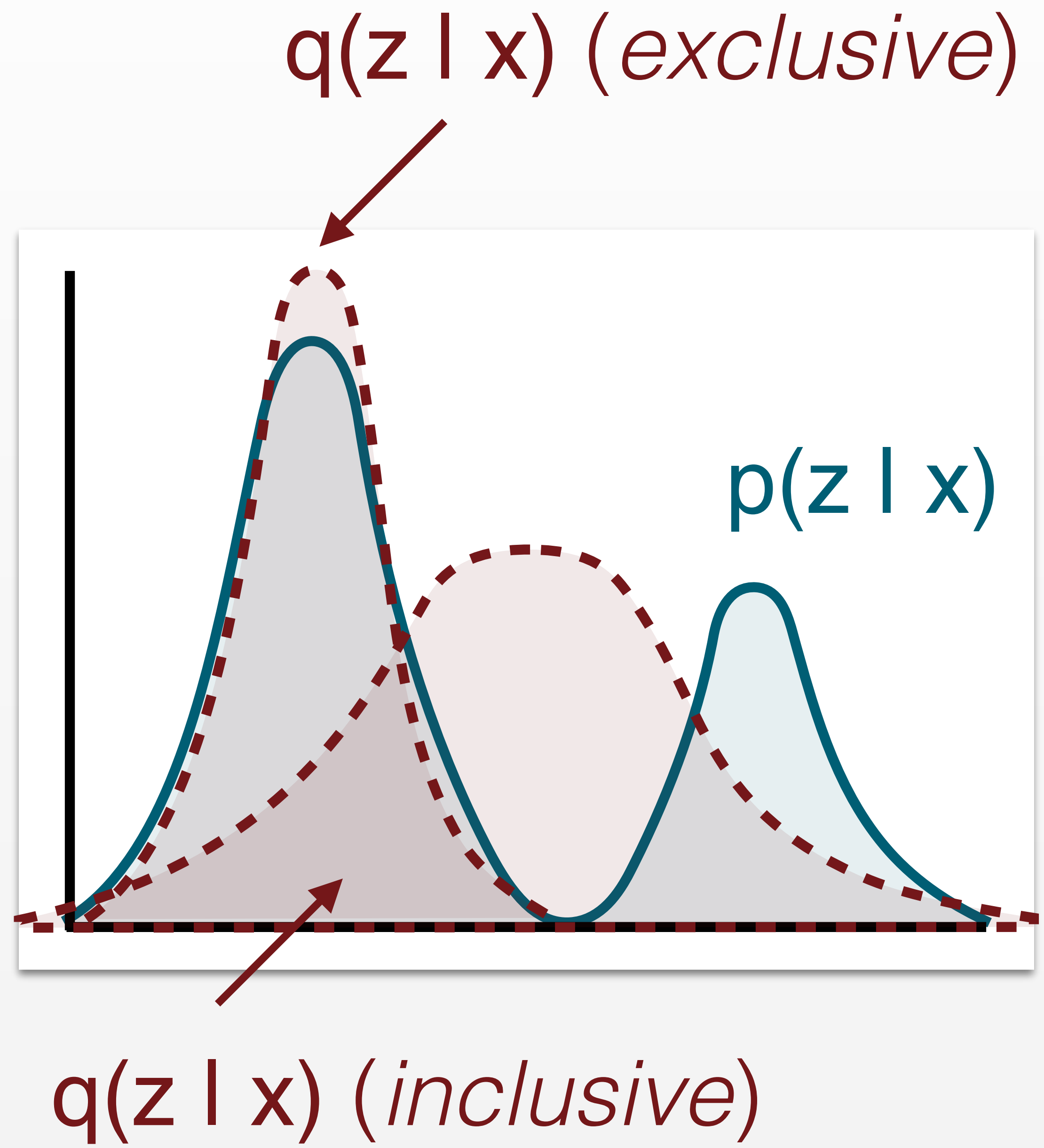
Minimizing the Inclusive KL

Variational Autoencoders

$$\underset{\phi}{\operatorname{argmin}} \ \mathrm{KL}\big(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\big|\big|\, p_\theta(\boldsymbol{z} \mid \boldsymbol{x})\big)$$

Wake-Sleep / EP Methods

$$\underset{\phi}{\operatorname{argmin}} \ \mathrm{KL}\big(p_\theta(\boldsymbol{z} \mid \boldsymbol{x}) \,\big|\big|\, q_\phi(\boldsymbol{z} \mid \boldsymbol{x})\big)$$



Minimizing the Inclusive KL

Variational Autoencoders

$$-\nabla_\phi \,\mathrm{KL}\big(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,||\, p_\theta(\boldsymbol{z} \mid \boldsymbol{x})\big)$$

$$= -\nabla_\phi \,\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}{p_\theta(\boldsymbol{z} \mid \boldsymbol{x})}\right]$$

Difficult: Need to approximate expectation that depends on ϕ

Solution: reparameterization or REINFORCE-style estimators

Wake-Sleep / EP Methods

$$-\nabla_\phi \,\mathrm{KL}\big(p_\theta(\boldsymbol{z} \mid \boldsymbol{x}) \,||\, q_\phi(\boldsymbol{z} \mid \boldsymbol{x})\big)$$

$$= -\nabla_\phi \mathbb{E}_{p_\theta(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{z} \mid \boldsymbol{x})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right]$$

$$= \mathbb{E}_{p_\theta(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_\phi \log q_\phi(\boldsymbol{z} \mid \boldsymbol{x})\right]$$

Easier: Expectation depends on θ (generative parameters)

Solution: sample from $p_\theta(z \mid x)$

Reweighted Wake-sleep Style Methods

$$-\nabla_\phi \mathrm{KL}\big(p_\theta(\boldsymbol{z} \mid \boldsymbol{x}) \,\big|\big|\, q_\phi(\boldsymbol{z} \mid \boldsymbol{x})\big) = \mathbb{E}_{p_\theta(\boldsymbol{z}|\boldsymbol{x})}\Big[\nabla_\phi \log q_\phi(\boldsymbol{z} \mid \boldsymbol{x})\Big]$$

$$\nabla_\theta \log p_\theta(\boldsymbol{x}) = \mathbb{E}_{p_\theta(\boldsymbol{z}|\boldsymbol{x})}\Big[\nabla_\theta \log p_\theta(\boldsymbol{x}, \boldsymbol{z})\Big]$$

Approximate with *any* importance sampler
(lots of probabilistic programming methods available,
can use learned $q_\phi(z \mid x)$ as proposals)

Reweighted Wake-sleep Style Methods

Better gradient estimates

Importance Sampling

Use proposals $q_\phi(z \mid x)$
to sample from $p_\theta(z \mid x)$

Variational Inference

Learn proposals $q_\phi(z \mid x)$
using samples from $p_\theta(z \mid x)$

Better proposals

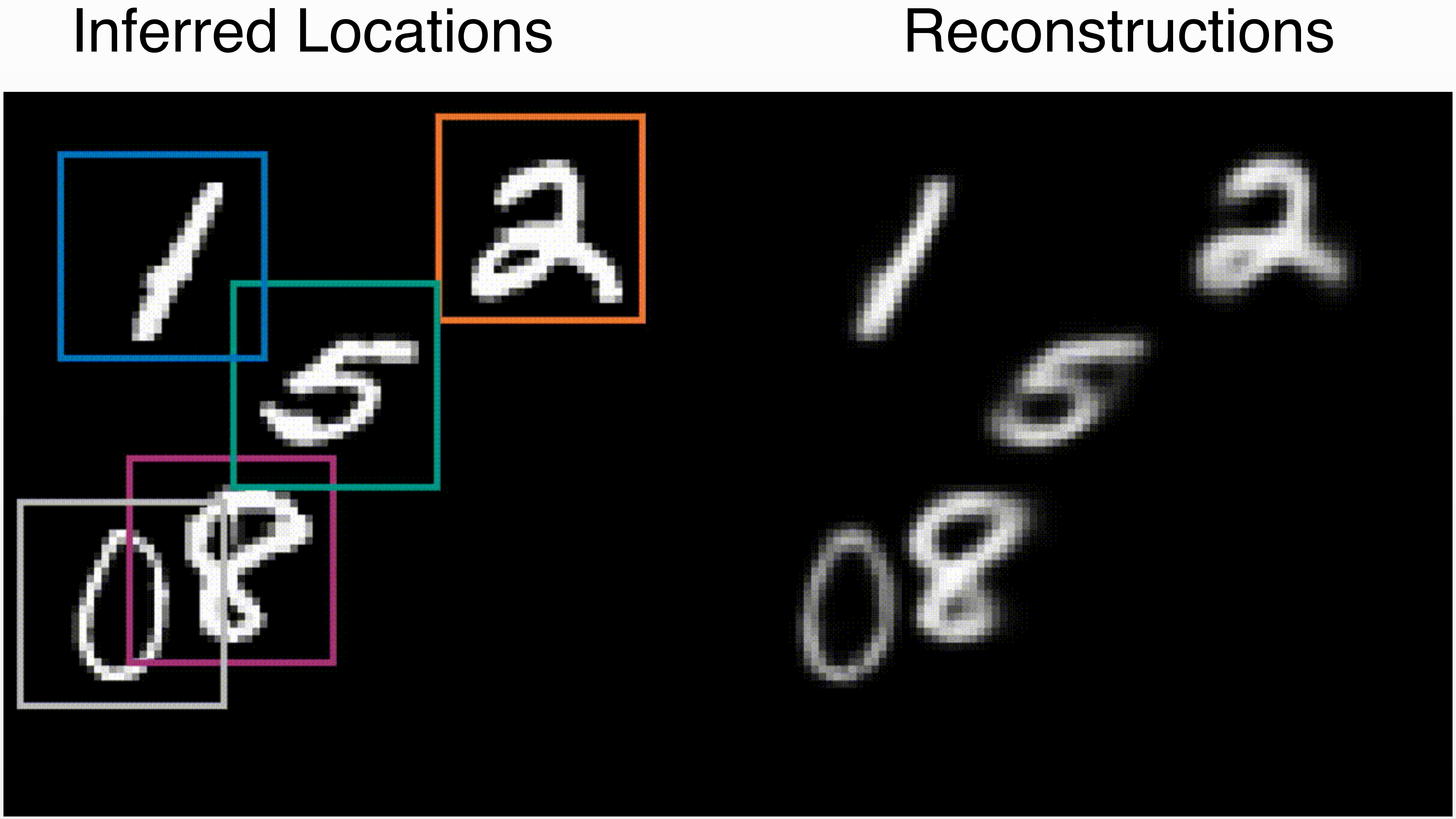
Inference methods from probabilistic programming

Automatic differentiation and neural networks

[Bornschein and Bengio, ICLR 2015]

[Le, Kosiorek, Siddarth, Teh, Wood, UAI 2019]

Example: Unsupervised Tracking



Fully unsupervised (learns sub-model for MNIST)
Scales to O(100) frames and ~5 digits (possibly more)

Thinking Compositionally

What (inference) DSL could define this sampler?

Algorithm 1 Amortized Population Gibbs Sampling

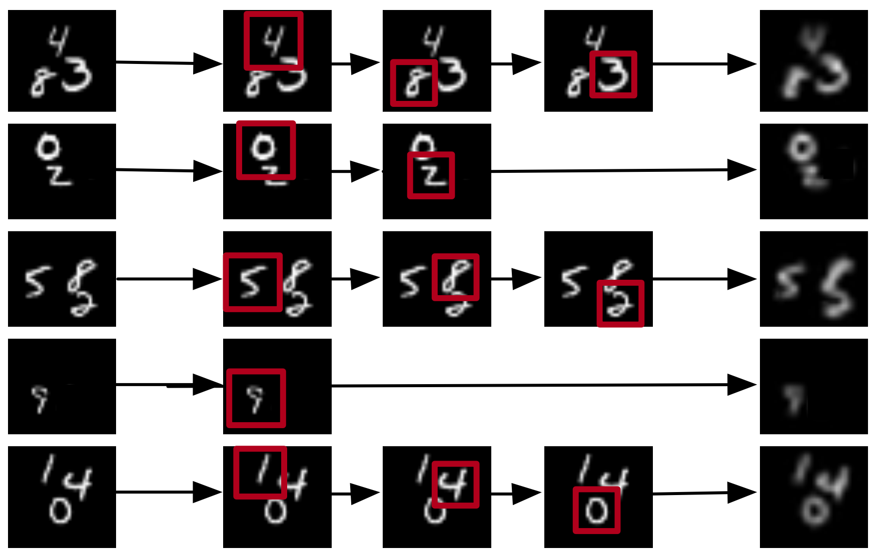
for n **in** $1, \ldots, N$ **do**
  $G_\phi = 0$
  $x^n \sim p^{\text{DATA}}(x)$
  for l **in** $1, \ldots, L$ **do**
    $z^{n,1,l} \sim q_\phi(z \mid x^n)$
    $w^{n,1,l} \leftarrow p_\theta(x^n, z^{n,1,l}) \,/\, q_\phi(z^{n,1,l})$
  for k **in** $2, \ldots, K$ **do**
    $\tilde{z}, \tilde{w} = z^{n,k-1}, w^{n,k-1}$
    for b **in** $1, \ldots, B$ **do**
      $\tilde{z}, \tilde{w} = \text{RESAMPLE}(\tilde{z}, \tilde{w})$
      for l **in** $1, \ldots, L$ **do**
        $\tilde{z}'^{\,l}_b \sim q_\phi(\cdot \mid x^n, \tilde{z}^l_{-b})$
        $\tilde{w}^l = \frac{p_\theta(x^n, \tilde{z}'^{\,l}_b, \tilde{z}^l_{-b})\, q_\phi(\tilde{z}^l_b \mid x^n, \tilde{z}^l_{-b})}{p_\theta(x^n, \tilde{z}^l_b, \tilde{z}^l_{-b})\, q_\phi(\tilde{z}'^{\,l}_b \mid x^n, \tilde{z}^l_{-b})}\, \tilde{w}^l$
        $\tilde{z}^l_b = \tilde{z}'^{\,l}_b$
      $G_\phi = G_\phi + \sum_{l=1}^{L} \frac{\tilde{w}^l}{\sum_{l'} \tilde{w}^{l'}} \frac{d}{d\phi} \log q_\phi(\tilde{z}^l_b \mid x^n, \tilde{z}^l_{-b})$
    $z^{n,k}, w^{n,k} = \tilde{z}, \tilde{w}$
return G_ϕ, z, w ▷ Output: Grac

- APG combines inference from probabilistic programming with SGD-based methods
- Known building blocks (SMC and RWS), but not trivial to combine correctly
- Can we define compositional methods for importance sampling and gradient estimation?

[Wu, Zimmermann, Sennesh, Le, van de Meent, arXiv 2019]

Static vs Dynamic Models

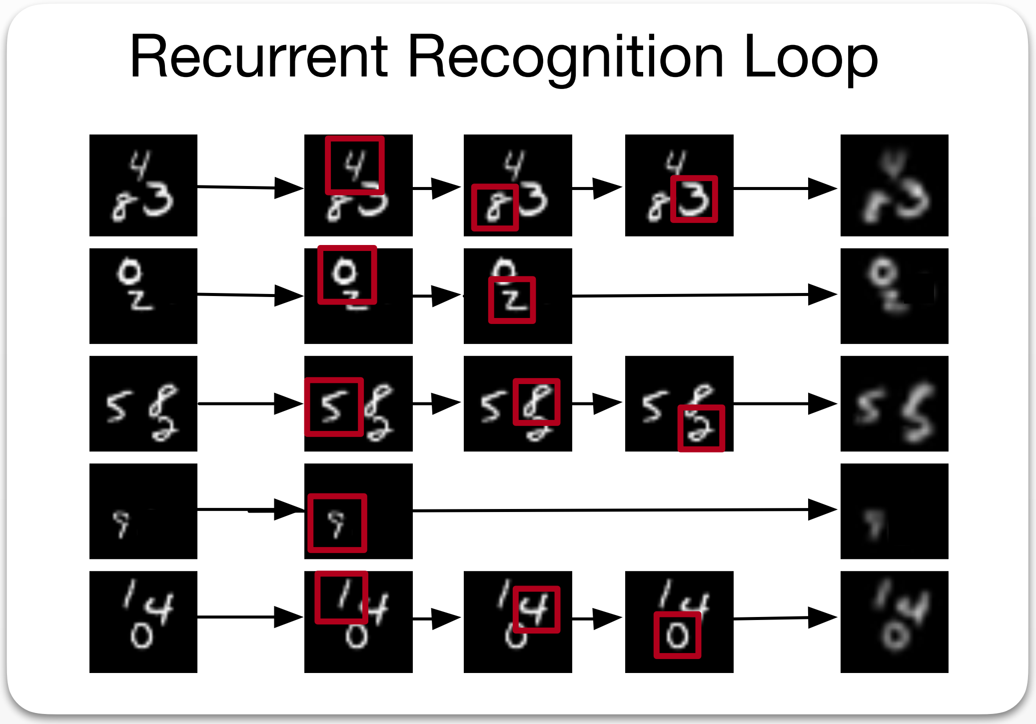
Recurrent Recognition Loop



```
def next_object(image, objects):
  area = detect(image, objects)
  features = recognize(image, area)
  return area, features
```

```
def detect_objects(image, canvas):
  objects = []
  while not similar(image, objects):
   objects.append(
      next_object(image, canvas))
 return objects, canvas
```

Static vs Dynamic Models

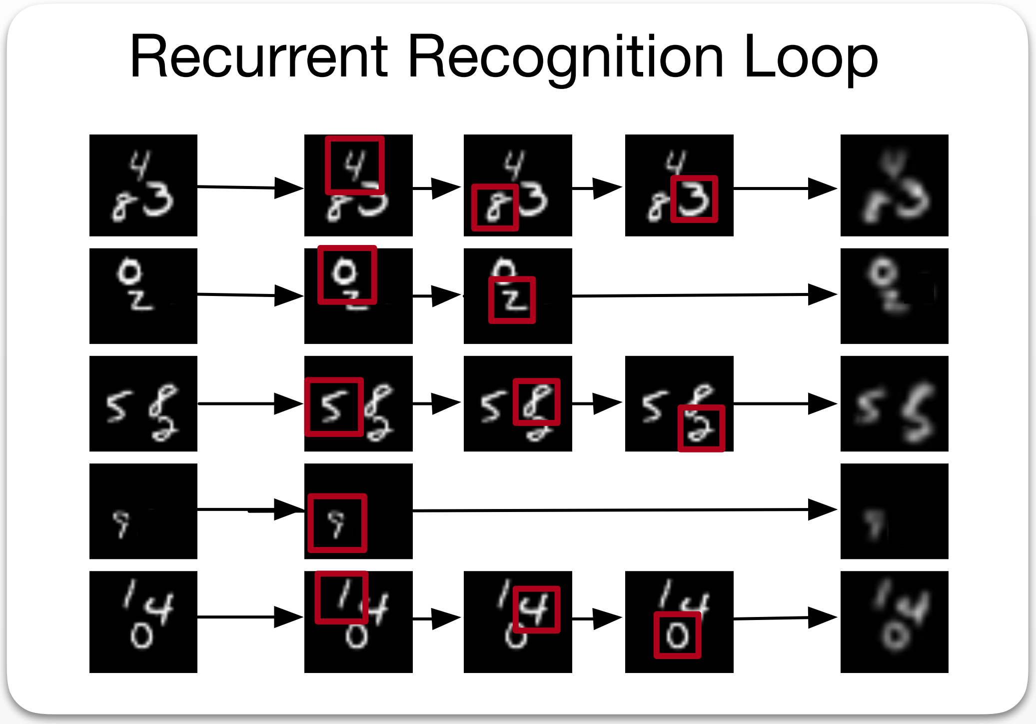



```
def next_object(image, objects):
  area = detect(image, objects)
  features = recognize(image, area)
  return area, features

def detect_objects(image, canvas):
  objects = []
  while not similar(image, objects):
   objects.append(
       next_object(image, canvas))
 return objects, canvas
```

Dynamic Computation Graphs: Number of variable nodes is data-dependent and/or stochastic

Static vs Dynamic Models




```
def next_object(image, objects):
  area = detect(image, objects)
  features = recognize(image, area)
  return area, features

def detect_objects(image):
  objects = []
  for k in range(3):
    object =
      next_object(image, canvas)
    if not similar(image, objects):
     objects.append(object)
return objects
```

Static Computation Graphs: Set of nodes and dependency graph determinable from static analysis.

Static vs Dynamic Models

	Static Graphs	Dynamic Graphs
Prob. Prog.	Infer.NET, Stan*	Venture, Turing, Gen, WebPPL, Anglican, Birch
Deep Learning	TensorFlow**	PyTorch, mxnet

(*dynamic graphs, static support)

(**in non-eager mode)

Programs as Importance Samplers

Output Trace Weight Program Address Input

$$z, \xi, w \leftsquigarrow f(\alpha, x)$$

$$\gamma_f(\xi\,;x) = \omega_f(\xi, x)\, p_f(\xi\,;x) \qquad z = \zeta_f(\xi, x)$$

Unnormalized Density

Conditioning Factor

Prior

Output Computation

$$\xi = [\alpha_1 \mapsto x_1, \ldots, \alpha_N \mapsto x_N]$$

Address Value

Programs as Importance Samplers

Output Trace Weight Program Address Input

$$z, \xi, w \leftsquigarrow f(\alpha, x)$$

$$\gamma_f(\xi\,; x) = \omega_f(\xi, x)\, p_f(\xi\,; x) \qquad z = \zeta_f(\xi, x)$$

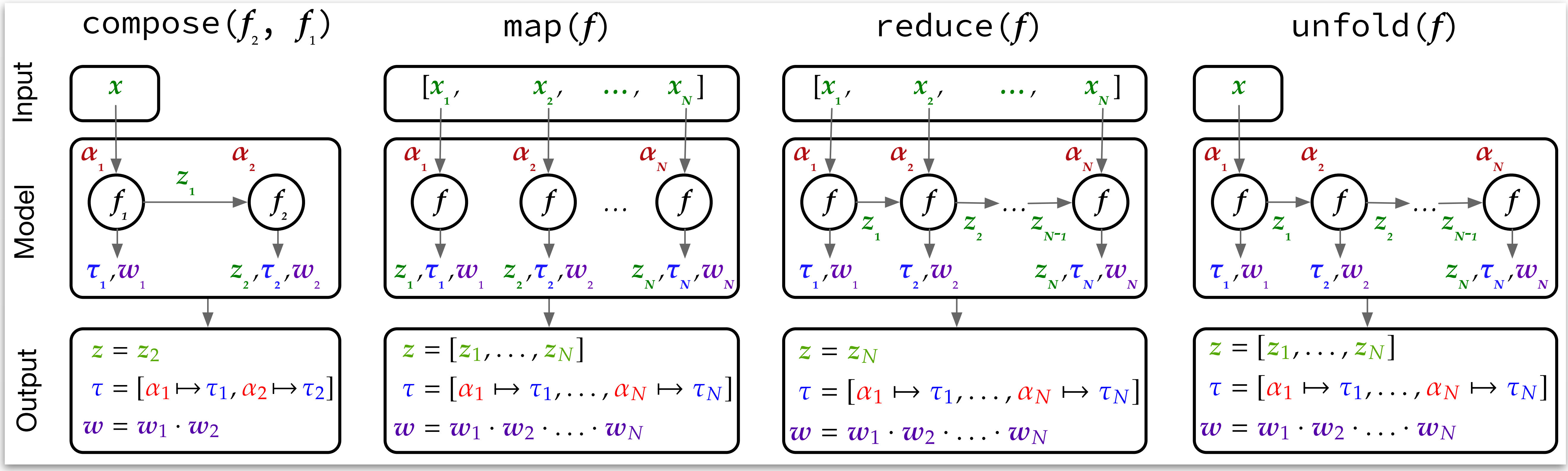
(measure semantics)

$$\xi \sim p_f(\cdot\,; x) \qquad w = \omega_f(\xi; x)$$

(likelihood weighting semantics)

[Ścibior, Kammar, Vákár, Staton, Yang, Cai, Ostermann, Heunen, Gharamani POPL 2018]

Model Combinators

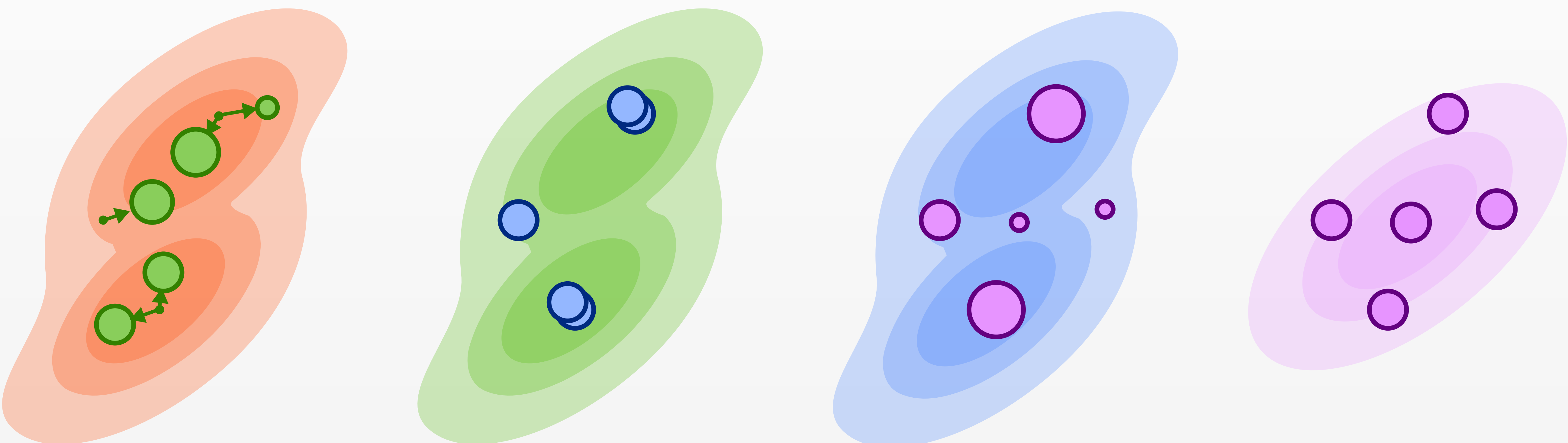


Idea: *Static* graphs with *dynamic* complexity

[Sennesh, Ścibior, Wu, van de Meent, (*arXiv*)]

Sampling Primitives

3 Building Blocks for Importance Samplers



Move *Resample* *Propose*

Inference Combinators

	`propose(f, g)`	`move(f, g)`	`resample(f)`
Input	x	x	x
Sampler	$z_g, \tau_g, w_g \leftsquigarrow g(x)$ $z_f, \tau_f, w_f \leftsquigarrow f(x; \tau_g)$	$z_f, \tau_f, w_f \leftsquigarrow f(x; \tau_g)$ $z_g, \tau_g, w_g \leftsquigarrow g(x, z_f)$ $z'_f, \tau'_f, w'_f \leftsquigarrow f(x; \tau_g)$	$z_k, \tau_k, w_k \leftsquigarrow f(x) \quad k = 1, \ldots, K$ $a_k \sim \text{Discrete}(w_1, \ldots, w_K)$
Output	$z = z_f \qquad \tau = \tau_f$ $w = \dfrac{p_g(\tau_g; \tau_f, x)}{\gamma_g(\tau_g; x)} \cdot w_f \cdot w_g$	$z = z'_f \qquad \tau = \tau'_f$ $w = \dfrac{p_g(\tau_g; \tau_f, x)}{\gamma_g(\tau_g; x, z_f)} \cdot w_f \cdot w_g \cdot w'_f$	$z_k = z_{a_k} \qquad \tau_k = \tau_{a_k}$ $w_k = \dfrac{1}{K} \sum_{k=1}^{K} w_k$

Idea: Change the evaluation (sampling) strategy, whilst leaving the target density unaffected.

[Sennesh, Ścibior, Wu, van de Meent, (*arXiv*)]

Dealing with Missing Variables

Target (variables: v, x1, x2)

```
def f(x0):
  x1 = sample(categorical(prior_1), name='x1')
  x2 = sample(normal(prior_2), name='x2')
  v = sample(normal(dec_v(x1, x2)), name='v')
  observe(normal(dec_0(v)), x0, name='x0')
```

$$\gamma_f(x_1, x_2, v\,; x_0) = p_f(x_0 \mid v)\, p_f(v \mid x_1, x_2)\, p_f(x_1, x_2)$$

Proposal (variables: u, x1, x2)

```
def g(x0):
  u = sample(normal(enc_u(x0)), name='u')
  x1 = sample(categorical(enc_1(u)), name='x1')
  x2 = sample(normal(enc_2(u)), name='x2')
```

$$p_g(x_1, x_2, u\,; x_0) = p_g(x_1, x_2 \mid u)\, p_g(u\,; x_0)$$

Dealing with Missing Variables

Extended Target (variables: v, x1, x2, u)

```
def f(x0):
  x1 = sample(categorical(prior_1), name='x1')
  x2 = sample(normal(prior_2), name='x2')
  v = sample(normal(dec_v(x1, x2)), name='v')
  observe(normal(dec_0(v)), x0, name='x0')
```

$$\tilde{\gamma}_f(x_1, x_2, v\,; x_0) = p_f(x_0 \mid v)\, p_f(v \mid x_1, x_2)\, p_f(x_1, x_2)\, p_g(u\,; x_0)$$

Extended Proposal (variables: u, x1, x2, v)

```
def g(x0):
  u = sample(normal(enc_u(x0)), name='u')
  x1 = sample(categorical(enc_1(u)), name='x1')
  x2 = sample(normal(enc_2(u)), name='x2')
```

$$\tilde{p}_g(x_1, x_2, u\,; x_0) = p_g(x_1, x_2 \mid u)\, p_g(u\,; x_0)\, p_f(v \mid x_1, x_2)$$

Dealing with Missing Variables

Extended Target (variables: v, x1, x2, u)

```
def f(x0):
  x1 = sample(categorical(prior_1), name='x1')
  x2 = sample(normal(prior_2), name='x2')
  v = sample(normal(dec_v(x1, x2)), name='v')
  observe(normal(dec_0(v)), x0, name='x0')
```

$$\tilde{\gamma}_f(x_1, x_2, v\,; x_0) = p_f(x_0 \mid v)\, p_f(v \mid \cancel{x_1, x_2})\, p_f(x_1, x_2)\, p_g(\cancel{u\,; x_0})$$

Extended Proposal (variables: u, x1, x2, v)

```
def g(x0):
  u = sample(normal(enc_u(x0)), name='u')
  x1 = sample(categorical(enc_1(u)), name='x1')
  x2 = sample(normal(enc_2(u)), name='x2')
```

$$\tilde{p}_g(x_1, x_2, u\,; x_0) = p_g(x_1, x_2 \mid u)\, p_g(\cancel{u\,; x_0})\, p_f(v \mid \cancel{x_1, x_2})$$

Dealing with Missing Variables

$$w_f = \frac{\tilde{\gamma}_f(x_1, x_2, v\,; x_0)}{\tilde{p}_g(x_1, x_2, u\,; x_0)} = \frac{p_f(x_0 \mid v)\, p_f(x_1, x_2)}{p_g(x_1, x_2 \mid u)}$$

(ratio for variables common to <u>both</u> models)

Proposal

Conditioned Target

$$\frac{z_1, \xi_1, w_1 \leftsquigarrow g(x_0) \qquad z_2, \xi_2, w_2 \leftsquigarrow f(x_0\,; \xi_1)}{z_2,\ \xi_2,\ w_1 \cdot w_2 \cdot \dfrac{p_g(\xi_1\,;\, \xi_2, x_0)}{\gamma_g(\xi_1\,;\, x_0)} \leftsquigarrow \texttt{propose}(f, g)(x_0)}$$

Conditioned reverse

Combinator

Model Combinators

define compositional model structure

Function Composition

$$\frac{z_1, \tau_1, w_1 \leftsquigarrow g\,(x_0\,;\tau_0(\alpha_1)) \qquad z_2, \tau_2, w_2 \leftsquigarrow f\,(z_1\,;\tau_0(\alpha_2))}{z_2, [\alpha_1 \mapsto \tau_1, \alpha_2 \mapsto \tau_2],\;\; w_1 \cdot w_2 \leftsquigarrow \texttt{compose}(f\,, g)\,(x_0\,;\tau_0)}$$

Map over Sequence

$$\frac{z_n, \tau_n, w_n \leftsquigarrow f\,(x_n\,;\tau_0(\alpha_n)) \qquad \text{for } n = 1, \ldots, N}{(z_1, \ldots, z_N), [\alpha_1 \mapsto \tau_1, \ldots, \alpha_N \mapsto \tau_N], \prod_{n=1}^{N} w_n \leftsquigarrow \texttt{map}(f)\,((x_1, \ldots, x_N)\,;\tau_0)}$$

Reduce / Fold over Sequence

$$\frac{z_1, \tau_1, w_1 \leftsquigarrow g\,(x_0\,;\tau_0(\alpha_1)) \qquad z_n, \tau_n, w_n \leftsquigarrow f\,(z_{n-1}\,;\tau_0(\alpha_n)) \qquad \text{for } n = 2, \ldots, N}{z_N, [\alpha_1 \mapsto \tau_1, \ldots, \alpha_N \mapsto \tau_N], \prod_{n=1}^{N} w_n \leftsquigarrow \texttt{reduce}(f\,, g)\,((x_1, \ldots, x_N)\,;\tau_0)}$$

[Sennesh, Ścibior, Wu, van de Meent, (*arXiv*)]

Inference Combinators

define a compositional sampling strategy

Nested Importance Sampling

$$\frac{z_1, \tau_1, w_1 \leftsquigarrow g(x_0) \qquad z_2, \tau_2, w_2 \leftsquigarrow f(x_0\,; \tau_1)}{z_2,\ \ \tau_2,\ \ \dfrac{w_1 \cdot w_2}{\omega_g(\tau_1; \tau_2, x_0)} \ \leftsquigarrow\ \texttt{propose}(f, g)(x_0)}$$

Importance Resampling

$$\frac{z^k, \tau^k, w^k \leftsquigarrow f(\alpha, x) \quad \text{for } k = 1, \ldots, K \qquad a^1, \ldots, a^K \sim \text{Discrete}\left(w^1, \ldots, w^K\right)}{z^{a^k}, \tau^{a^k}, \frac{1}{K}\sum_k w^k \leftsquigarrow \texttt{resample}(f, K)(\alpha, x) \qquad \text{for } k = 1, \ldots, K}$$

Application of a Transition Kernel

$$\frac{z_1, \tau_1, w_1 \leftsquigarrow f(x_0) \qquad z_2, \tau_2, w_2 \leftsquigarrow g(z_1) \qquad z_3, \tau_3, w_3 \leftsquigarrow f(x_0\,; \tau_2)}{z_3,\ \ \tau_3,\ \ w_1 \cdot \dfrac{w_2 \cdot w_3}{\omega_g(\tau_2; \tau_3, z_1)} \ \leftsquigarrow\ \texttt{move}(f, g)(x_0)}$$

[Sennesh, Ścibior, Wu, van de Meent, (*arXiv*)]

Scaling up Amortized Inference

Deep Generative Model
program $p_\theta(x, z)$

Inference Model
program $q_\phi(z \mid x)$

Better gradient estimates

Importance Sampling

Use proposals $q_\phi(z \mid x)$
to sample from $p_\theta(z \mid x)$

Variational Inference

Learn proposals $q_\phi(z \mid x)$
using samples from $p_\theta(z \mid x)$

Better proposals

User-specified
importance sampler
(inference combinators)

User-specified
variational objective
(next talk)

Thank You!


Hao Wu


Eli Sennesh


Sam Stites


Heiko Zimmermann


Babak Esmaeli


Alican Bozkurt

Composing Modeling and Inference Operations with Probabilistic Program Combinators
E. Sennesh, A. Scibior, H. Wu, J.-W. van de Meent
ArXiv 2018 [https://arxiv.org/abs/1811.05965]

Amortized Gibbs Samplers with Neural Sufficient Statistics
H. Wu, H. Zimmermann, E. Sennesh, J.-W. Van de Meent
ArXiv 2019 [https://arxiv.org/abs/1911.01382]

Structured Neural Topic Models for Reviews
B. Esmaeli, H. Huang, B. Wallace, J.-W. van de Meent
AISTATS 2019 [http://proceedings.mlr.press/v89/esmaeili19b.html]

An Introduction to Probabilistic Programming
J-W van de Meent, B. Paige, H. Yang, F. Wood
ArXiv 2018 [https://arxiv.org/abs/1809.10756]

Evaluating Combinatorial Generalization in VAEs
A. Bozkurt, B. Esmaeili, D. H. Brooks, J. Dy, J.-W. van de Meent
ArXiv 2019 [https://arxiv.org/abs/1911.04594]

Source Code (Apache 2.0)
https://github.com/probtorch/probtorch

3M intel NSF DARPA