Inference Compilation

Tuan Anh Le
tuananh@robots.ox.ac.uk

(slides mostly borrowed from Frank Wood)

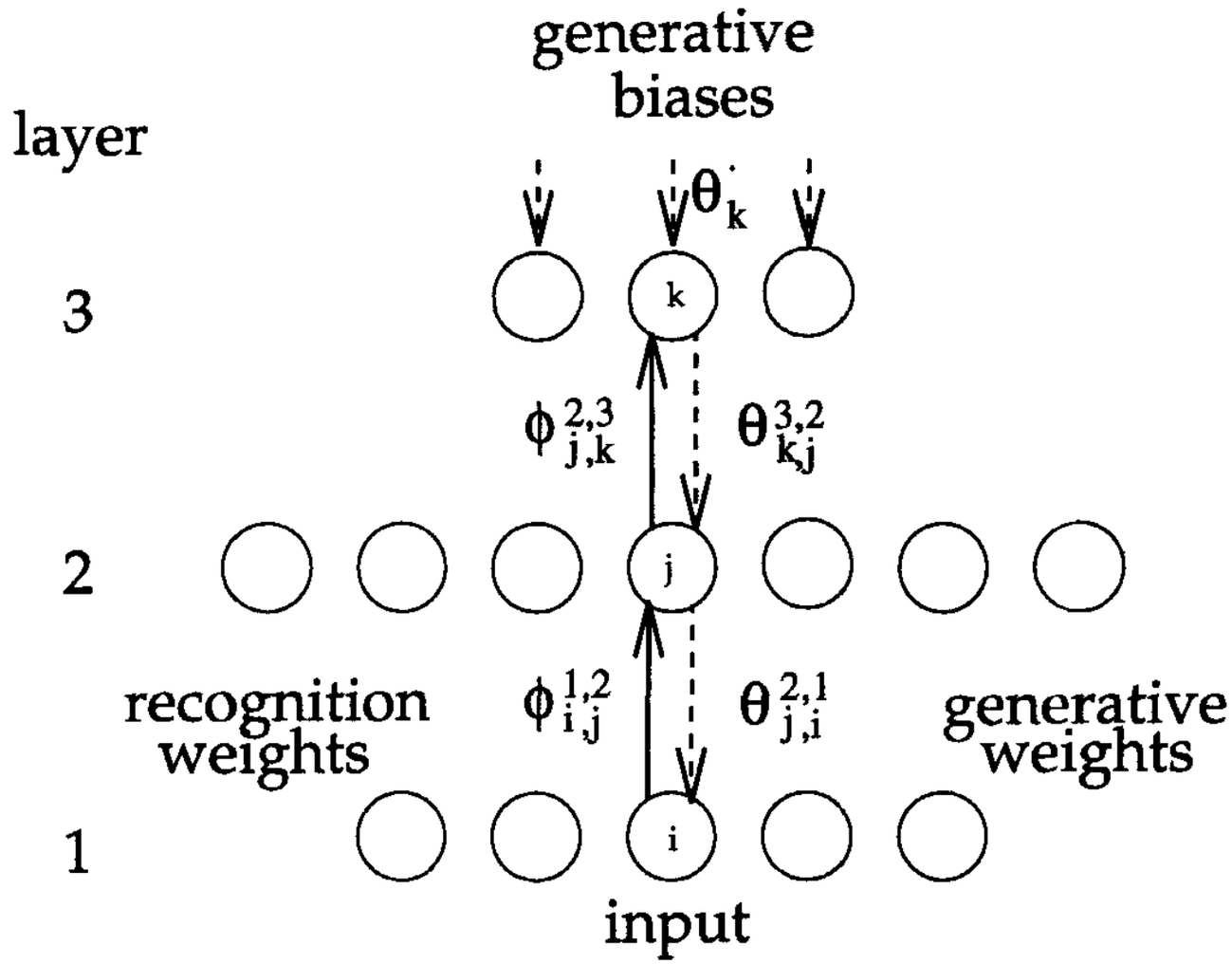
Accelerating Data Science with HPC Summer School
Lugano, September 4 – 6, 2017

Problem

"**Bayesian inference is computationally expensive**. Even approximate, sampling-based algorithms tend to take many iterations before they produce reasonable answers. In contrast, human recognition of words, objects, and scenes is extremely rapid, often taking only a few hundred milliseconds —only enough time for a **single pass from perceptual evidence to deeper interpretation**. Yet human perception and cognition are often well-described by **probabilistic inference in complex models**. How can we reconcile the speed of recognition with the expense of coherent probabilistic inference? How can we **build systems**, for applications like robotics and medical diagnosis, **that exhibit similarly rapid performance** at challenging inference tasks?"

Stuhlmüller A, Taylor J, Goodman N. Learning stochastic inverses. In Advances in Neural Information Processing Systems 2013 (pp. 3048-3056).

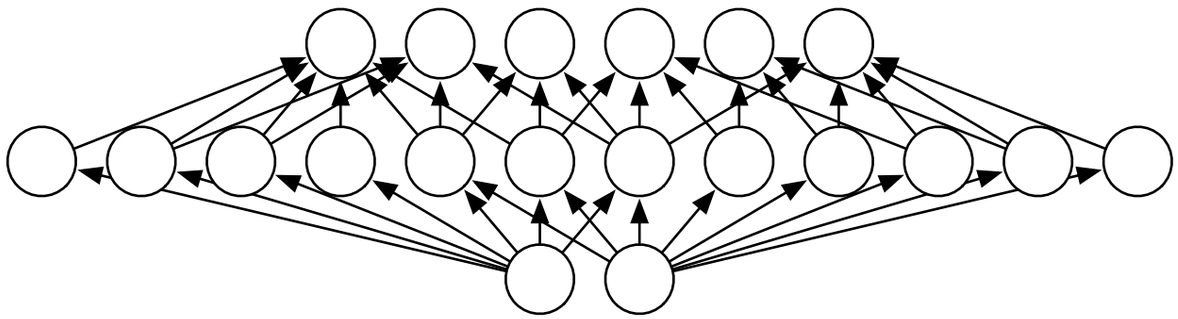
Helmholtz Machine



Dayan P, Hinton GE, Neal RM, Zemel RS. The Helmholtz machine. Neural computation. 1995 Sep;7(5):889-904.

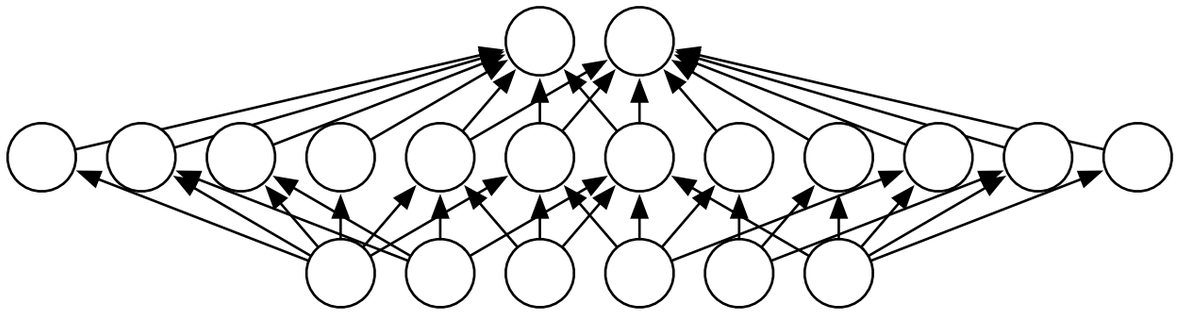
Variational Autoencoder - Neural Net View

$\mathbf{y}$ $p_\theta(\mathbf{y}|\mathbf{x})$



Decoder
Model

$q_\varphi(\mathbf{x}|\mathbf{y}) \approx p_\theta(\mathbf{x}|\mathbf{y}) \dashrightarrow \mathbf{x} \longleftarrow p_\theta(\mathbf{x})$



Encoder
Compiled Inference Artifact

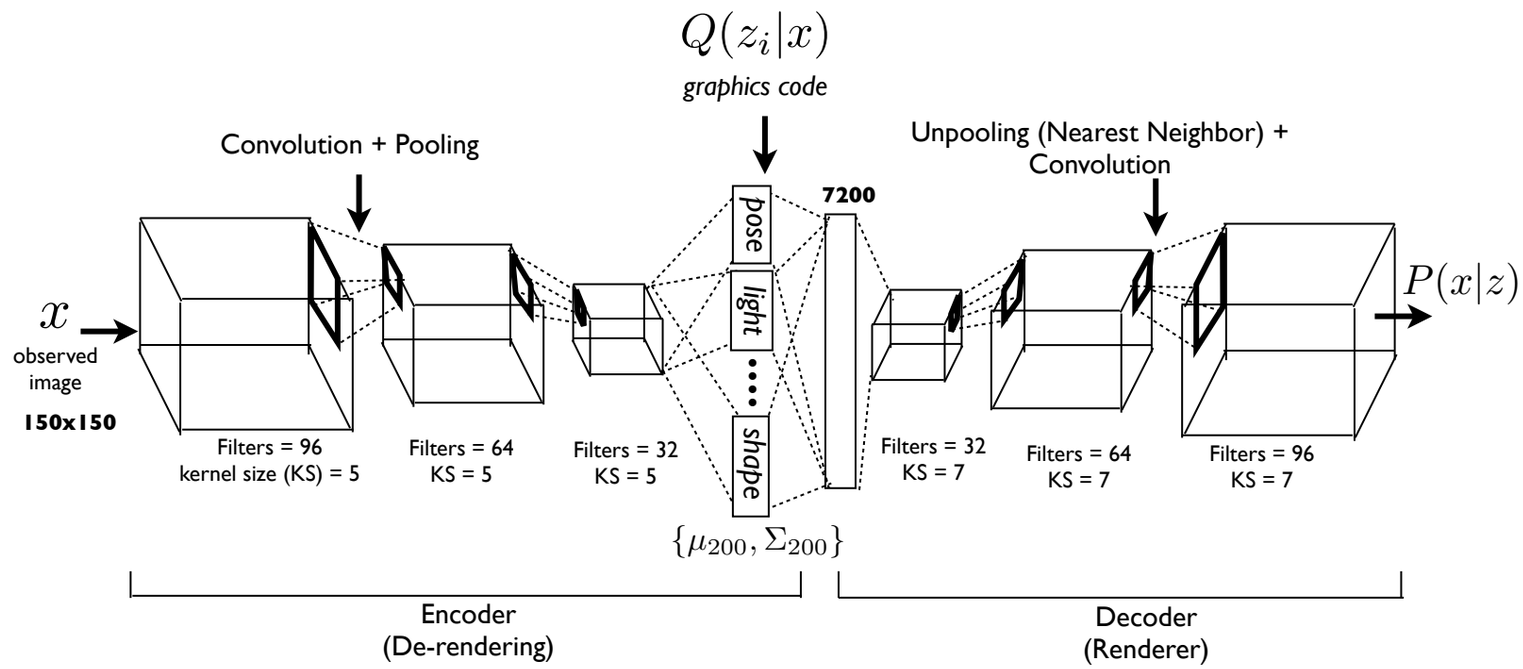
$\mathbf{y}$

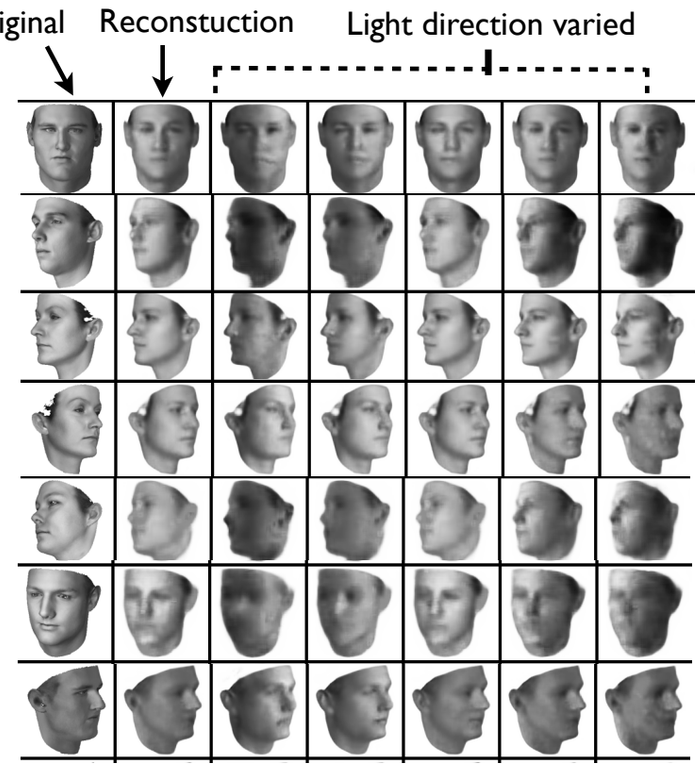
- Helmholtz machine realization with one layer of latents + reparameterization trick, diagonal Gaussian distributions, and multi-layer encoder and decoder neural networks, variational inference

Kingma DP, Welling M. Auto-encoding variational Bayes. arXiv preprint arXiv:1312.6114. 2013 Dec 20.

Structured Autoencoder Architectures

e.g. but many others





Posing and solving perception as inverse graphics requires

- Fast amortized/compiled inference
- Structured, interpretable model

Kulkarni TD, Whitney WF, Kohli P, Tenenbaum J. Deep convolutional inverse graphics network. In NIPS 2015 (pp. 2539-2547).

Inference Compilation: Desiderata

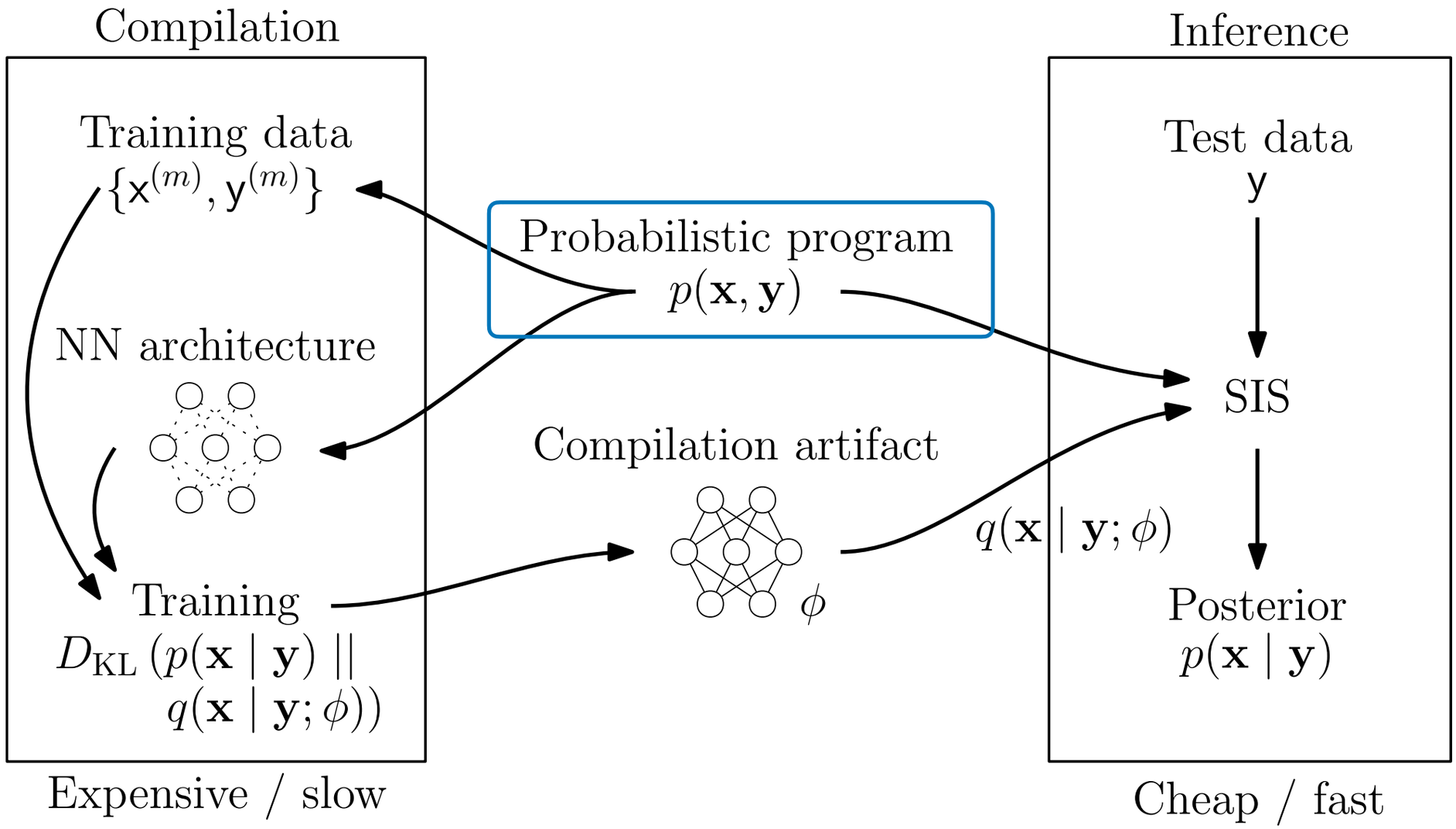
- Denote a model and inference problem as a probabilistic program

- “Compile” for hours or days, depending on the problem, CPU/GPUs at disposal, etc.

- Get a “compilation artifact” controller that enables fast, repeated inference in the original model that is compatible with asymptotically exact inference

- Useful model-based submodules within end-to-end AI pipelines

Inference Compilation



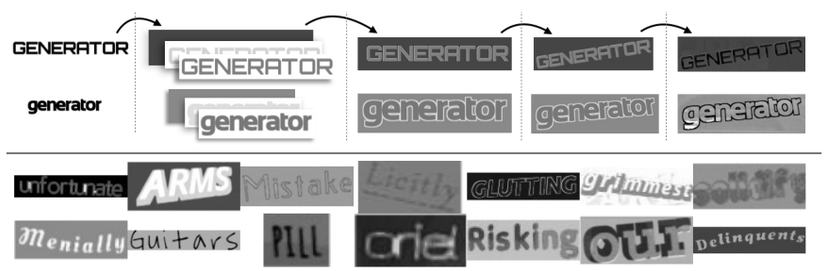
Input: an inference problem denoted in a probabilistic programming language (Anglican)

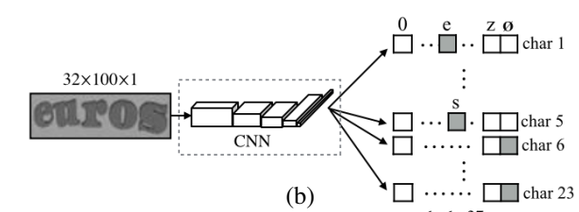
Output: a trained inference network (deep neural network “compilation artifact”)

Le TA, Baydin AG, Wood F. Inference Compilation and Universal Probabilistic Programming. AISTATS. 2017.

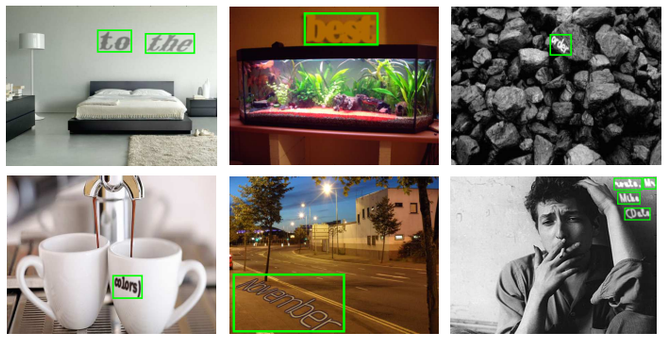
Synthetic Data for Training Deep Networks

Goodfellow, Bulatov, Ibarz, Arnoud, Shet; Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. 2014.





Jaderberg, Simonyan, Vedaldi, Zisserman; Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition. 2014.



Gupta, Vedaldi, Zisserman; Synthetic Data for Text Localisation in Natural Images. 2016.

Example: Graphical Model Inference

Graphical model with latents $\mathbf{x}$ and observations $\mathbf{y}$:

$$p(\mathbf{x}, \mathbf{y}) = \prod_{n=1}^{N} p(x_n | \mathrm{PA}(x_n)) \prod_{m=1}^{M} p(y_m | \mathrm{PA}(y_m))$$

Goal: Efficiently compute (approximation of):

- posterior $p(\mathbf{x}|\mathbf{y})$
- expectation under the posterior $\mathbb{E}_{p(\mathbf{x}|\mathbf{y})}[f(\mathbf{x})]$
- marginal likelihood $p(\mathbf{y})$

Importance sampling

For $k = 1, \ldots, K$:

Propose: $\mathbf{x}_k \sim q(\mathbf{x})$

Performance depends on quality of proposal

Weight: $w_k = \frac{p(\mathbf{x}_k, \mathbf{y})}{q(\mathbf{x}_k)}$

Normalize: $W_k = \frac{w_k}{\sum_{\ell=1}^{K} w_\ell}$

Approximate:

$$p(\mathbf{x}|\mathbf{y}) \approx \sum_{k=1}^{K} W_k \delta_{\mathbf{x}_k}(\mathbf{x})$$

$$\mathbb{E}_{p(\mathbf{x}|\mathbf{y})}[f(\mathbf{x})] \approx \sum_{k=1}^{K} W_k f(\mathbf{x}_k)$$

$$p(\mathbf{y}) \approx \sum_{k=1}^{K} w_k$$

Importance sampling

For $k = 1, \ldots, K$:

Propose: $\mathbf{x}_k \sim q(\mathbf{x}|\lambda)$

Weight: $w_k = \frac{p(\mathbf{x}_k, \mathbf{y})}{q(\mathbf{x}_k|\lambda)}$

Normalize: $W_k = \frac{w_k}{\sum_{\ell=1}^{K} w_\ell}$

Performance depends on quality of proposal

Adaptation

Approximate:

$$p(\mathbf{x}|\mathbf{y}) \approx \sum_{k=1}^{K} W_k \delta_{\mathbf{x}_k}(\mathbf{x})$$

$$\mathbb{E}_{p(\mathbf{x}|\mathbf{y})}[f(\mathbf{x})] \approx \sum_{k=1}^{K} W_k f(\mathbf{x}_k)$$

$$p(\mathbf{y}) \approx \sum_{k=1}^{K} w_k$$

One Notion of Optimal Proposal

Learning an importance sampling proposal for a single dataset

Target density $p(\mathbf{x}|\mathbf{y})$, approximating family $q(\mathbf{x}|\lambda)$

Single dataset $\mathbf{y}$: $\underset{\lambda}{\operatorname{argmin}}\, \mathrm{KL}(p(\mathbf{x}|\mathbf{y})||q(\mathbf{x}|\lambda))$ ← fit λ to learn an importance sampling proposal

$\underset{\lambda}{\operatorname{argmin}}\, \mathrm{KL}(q(\mathbf{x}|\lambda)||p(\mathbf{x}|\mathbf{y}))$ ← Note: opposite KL in VB/VAE

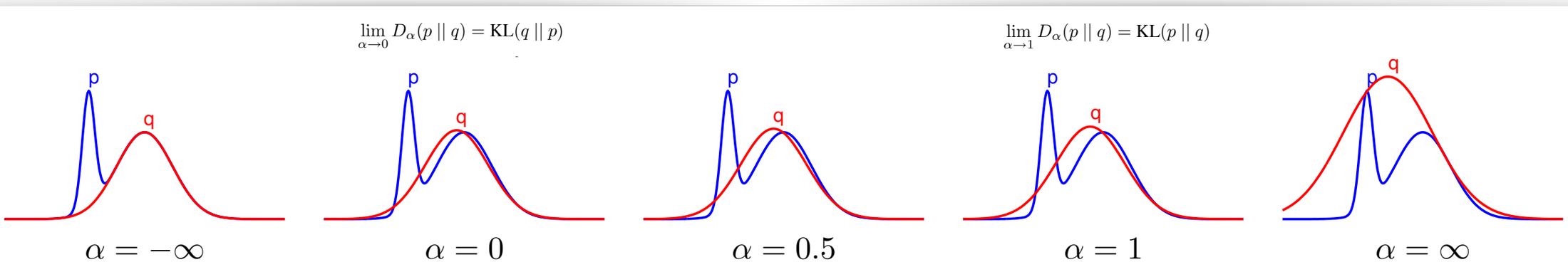


Figure 1: The Gaussian q which minimizes α-divergence to p (a mixture of two Gaussians), for varying α. $\alpha \to -\infty$ prefers matching one mode, while $\alpha \to \infty$ prefers covering the entire distribution.

Minka T. Divergence measures and message passing. Technical report, Microsoft Research; 2005 Dec 7.

Cornebise J, Moulines E, Olsson J. Adaptive methods for sequential importance sampling with application to state space models. *Statistics and Computing*, 18:461–480 (2008)

Compiling Away Runtime Costs of Inference

Learn to invert the generative model, before seeing data

Averaging over
all possible datasets: $\lambda = \varphi(\eta, \mathbf{y})$

$$\underset{\eta}{\operatorname{argmin}}\, \mathbb{E}_{p(\mathbf{y})} \left[\mathrm{KL}(p(\mathbf{x}|\mathbf{y}) || q(\mathbf{x}|\varphi(\eta, \mathbf{y}))) \right]$$

New objective function,
upper-level parameters:

$$\begin{aligned}
\mathcal{J}(\eta) &= \int p(\mathbf{y}) \mathrm{KL}(p(\mathbf{x}|\mathbf{y}) || q(\mathbf{x}|\lambda))\, \mathrm{d}\mathbf{y} \\
&= \int p(\mathbf{y}) \int p(\mathbf{x}|\mathbf{y}) \log \left[\frac{p(\mathbf{x}|\mathbf{y})}{q(\mathbf{x}|\varphi(\eta, \mathbf{y}))} \right] \mathrm{d}\mathbf{x}\, \mathrm{d}\mathbf{y} \\
&= \int \int p(\mathbf{x}, \mathbf{y}) \left[\log p(\mathbf{x}|\mathbf{y}) - \log q(\mathbf{x}|\varphi(\eta, \mathbf{y})) \right] \mathrm{d}\mathbf{x}\, \mathrm{d}\mathbf{y} \\
&= \mathbb{E}_{p(\mathbf{x},\mathbf{y})} \left[-\log q(\mathbf{x}|\varphi(\eta, \mathbf{y})) \right] + \text{const.}
\end{aligned}$$

Paige B, Wood F. Inference Networks for Sequential Monte Carlo in Graphical Models. ICML. JMLR W&CP 48: 3040-3049. 2016.

Compiling Away Runtime Costs of Inference

Learn to invert the generative model, before seeing data

Averaging over all possible datasets: $\lambda = \varphi(\eta, \mathbf{y})$

$$\underset{\eta}{\operatorname{argmin}}\, \mathbb{E}_{p(\mathbf{y})} \left[\mathrm{KL}(p(\mathbf{x}|\mathbf{y}) || q(\mathbf{x}|\varphi(\eta, \mathbf{y}))) \right]$$

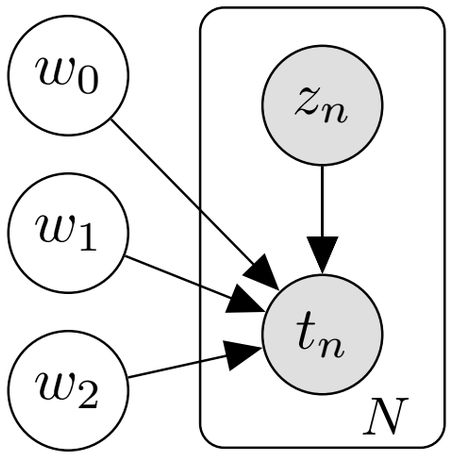
New objective function, upper-level parameters: $\mathcal{J}(\eta) = \mathbb{E}_{p(\mathbf{x},\mathbf{y})} \left[-\log q(\mathbf{x}|\varphi(\eta, \mathbf{y})) \right] + \text{const.}$

Can train using SGD with samples from the joint: $\nabla_\eta \mathcal{J}(\eta) = \mathbb{E}_{p(\mathbf{x},\mathbf{y})} \left[-\nabla_\eta \log q(\mathbf{x}|\varphi(\eta, \mathbf{y})) \right]$

Paige B, Wood F. Inference Networks for Sequential Monte Carlo in Graphical Models. ICML. JMLR W&CP 48: 3040-3049. 2016.

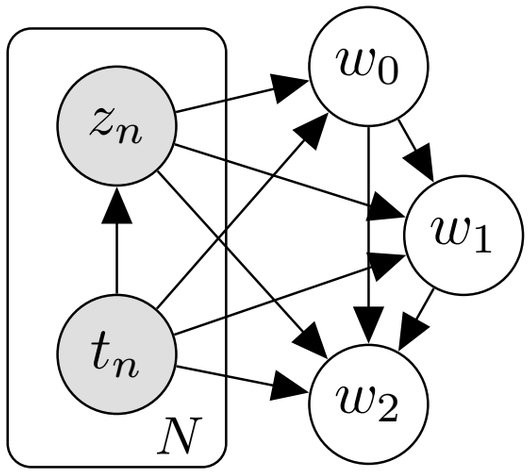
Example: Non-Conjugate Robust Polynomial Regression

Generative model



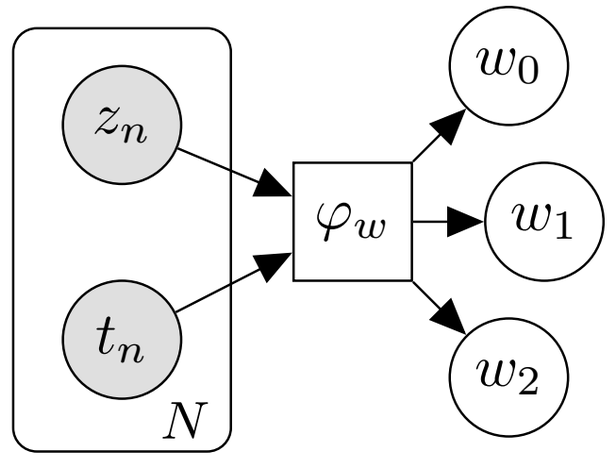
automatic

Inverted model



hand

Neural net proposal



$$w_d \sim \text{Laplace}(0, 10^{1-d}) \quad \text{for } d = 0, 1, 2;$$

$$t_n \sim \text{t}_\nu(w_0 + w_1 z_n + w_2 z_n^2, \epsilon^2) \quad \text{for } n = 1, \ldots, N$$

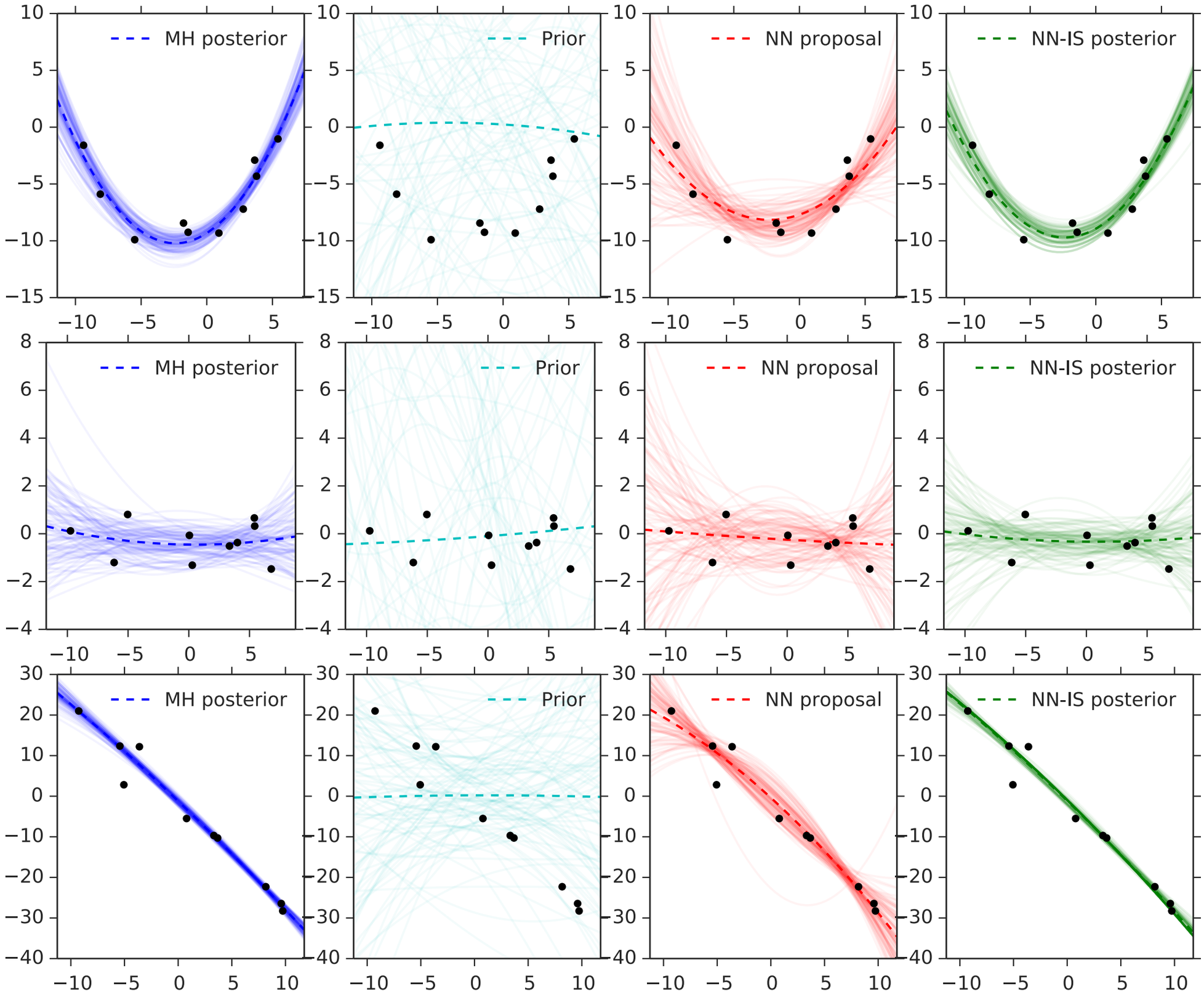
$$\nu = 4, \epsilon = 1, \text{and } z_n \in (-10, 10)$$

$$q(w_{0:2} | z_{1:N}, t_{1:N})$$

- Two layer MLP
- 200 units
- 3-component MOG for each output

Paige B, Wood F. Inference Networks for Sequential Monte Carlo in Graphical Models. ICML. JMLR W&CP 48: 3040-3049. 2016.

Example: Non-Conjugate Robust Polynomial Regression



Paige B, Wood F. Inference Networks for Sequential Monte Carlo in Graphical Models. ICML. JMLR W&CP 48: 3040-3049. 2016.

Inference Compilation for Higher-Order Languages

Le, T. A., Baydin, A. G., & Wood, F. (2017). Inference Compilation and Universal Probabilistic Programming. AISTATS (Vol. 54, pp. 1338–1348).

Higher-Order Probabilistic Programs

Execution trace of a probabilistic program

sample value — sample address — sample instance — trace length

$$(x_t, a_t, i_t)_{t=1}^{T}$$

Joint probability density

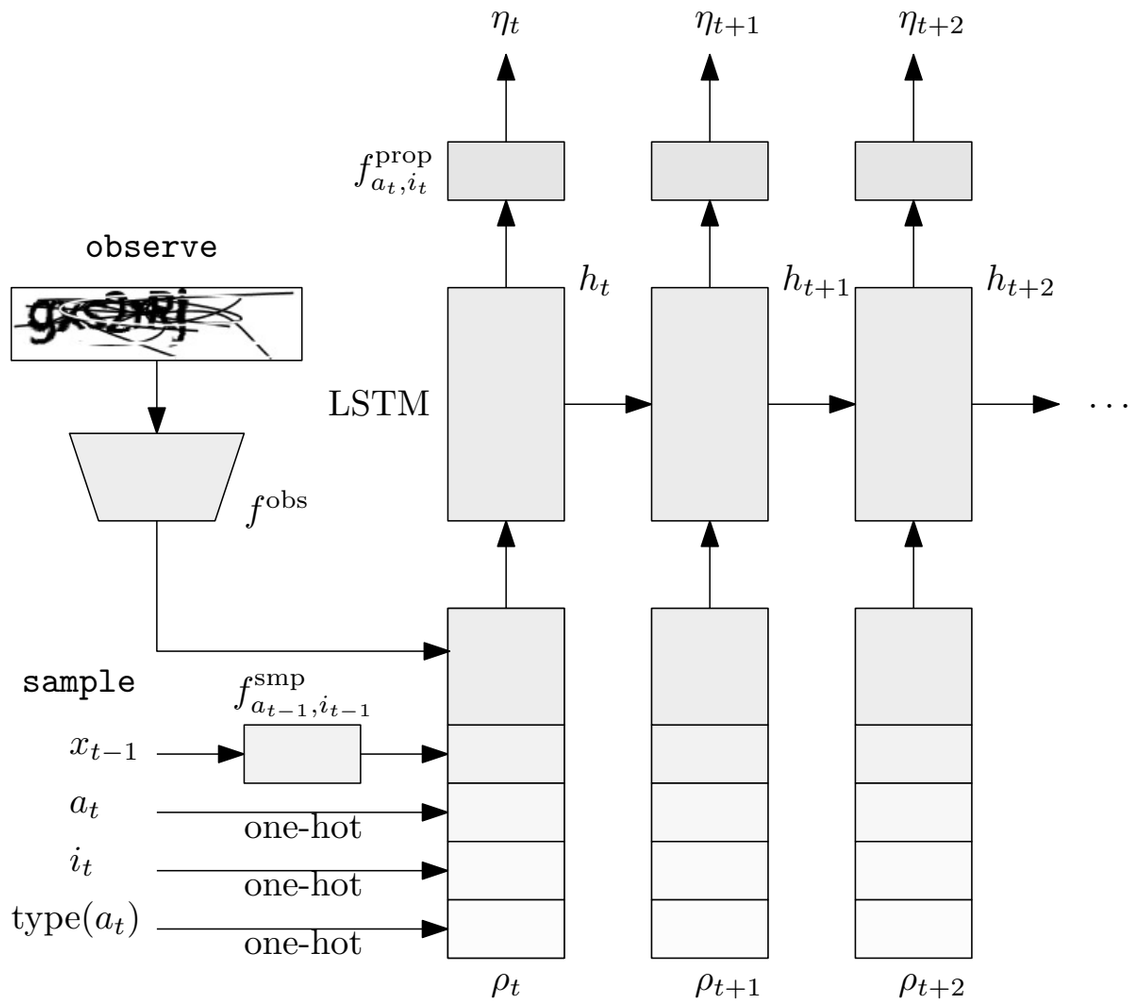
conditional priors — fixed observations — likelihoods

$$p(\mathbf{x}, \mathbf{y}) := \prod_{t=1}^{T} f_{a_t}(x_t | x_{1:t-1}) \prod_{n=1}^{N} g_n(y_n | x_{1:\tau(n)})$$

sample value — samples before x_t — nth observation — samples before nth observation

Proposal Neural Network Architecture

- A non-program-specific stacked LSTM core
- Proposal layers, observation and sample embeddings specified by the program
- **Compilation**: New layers are created and attached on-the-fly on first encounter with address–instance pairs
- **Inference**: The network is configured on-the-fly as the probabilistic program execution trace unravels



Hochreiter S, Schmidhuber J. Long short-term memory. Neural computation. 1997 Nov 15;9(8):1735-80.

Reed S, de Freitas N. Neural programmer-interpreters. arXiv preprint arXiv:1511.06279. 2015 Nov 19.

Le, T. A., Baydin, A. G., & Wood, F. (2017). Inference Compilation and Universal Probabilistic Programming. AISTATS (Vol. 54, pp. 1338–1348).

Example: Captcha

```
letters = []
num_letters = sample(UniformDiscrete(3, 10))
for i in range(num_letters):
    letters.append(sample(Uniform("a",…,"z","A",…,"Z")))

observe(render(letters), observed_captcha)
return letters
```

sample addresses

a_1

a_2

Importance sampling

For $k = 1, \ldots, K$:

Already trained

Propose: $\mathbf{x}_k \sim q(\mathbf{x}|\varphi(\eta, \mathbf{y}))$

Weight: $w_k = \frac{p(\mathbf{x}_k, \mathbf{y})}{q(\mathbf{x}_k|\lambda)}$

Normalize: $W_k = \frac{w_k}{\sum_{\ell=1}^{K} w_\ell}$

Approximate:

$$p(\mathbf{x}|\mathbf{y}) \approx \sum_{k=1}^{K} W_k \delta_{\mathbf{x}_k}(\mathbf{x})$$

$$\mathbb{E}_{p(\mathbf{x}|\mathbf{y})}[f(\mathbf{x})] \approx \sum_{k=1}^{K} W_k f(\mathbf{x}_k)$$

$$p(\mathbf{y}) \approx \sum_{k=1}^{K} w_k$$

Example: Captcha (Sampling from proposal)

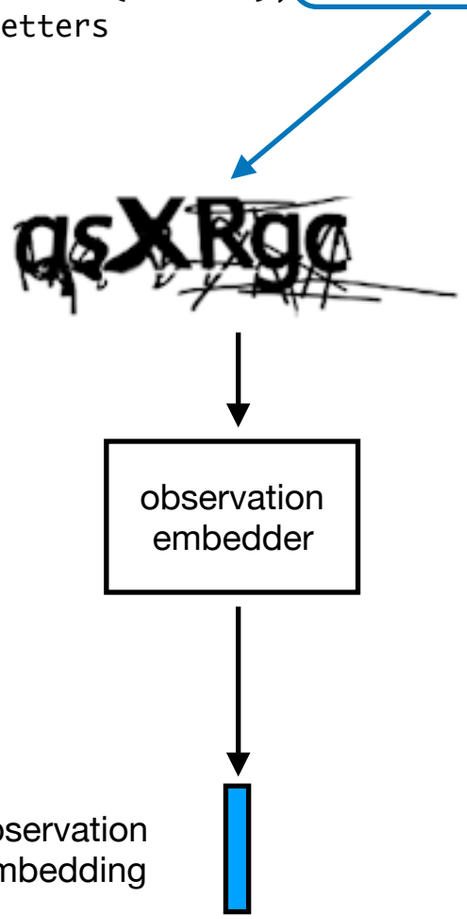
```
letters = []
num_letters = sample(UniformDiscrete(3, 10))
for i in range(num_letters):
   letters.append(sample(Uniform(“a”,…,”z”,“A”,…,”Z”)))

observe(render(letters), observed_captcha)
return letters
```

Example: Captcha (Sampling from proposal)

```
letters = []
num_letters = sample(UniformDiscrete(3, 10))
for i in range(num_letters):
   letters.append(sample(Uniform(“a”,…,”z”,“A”,…,”Z”)))

observe(render(letters), observed_captcha)
return letters
```

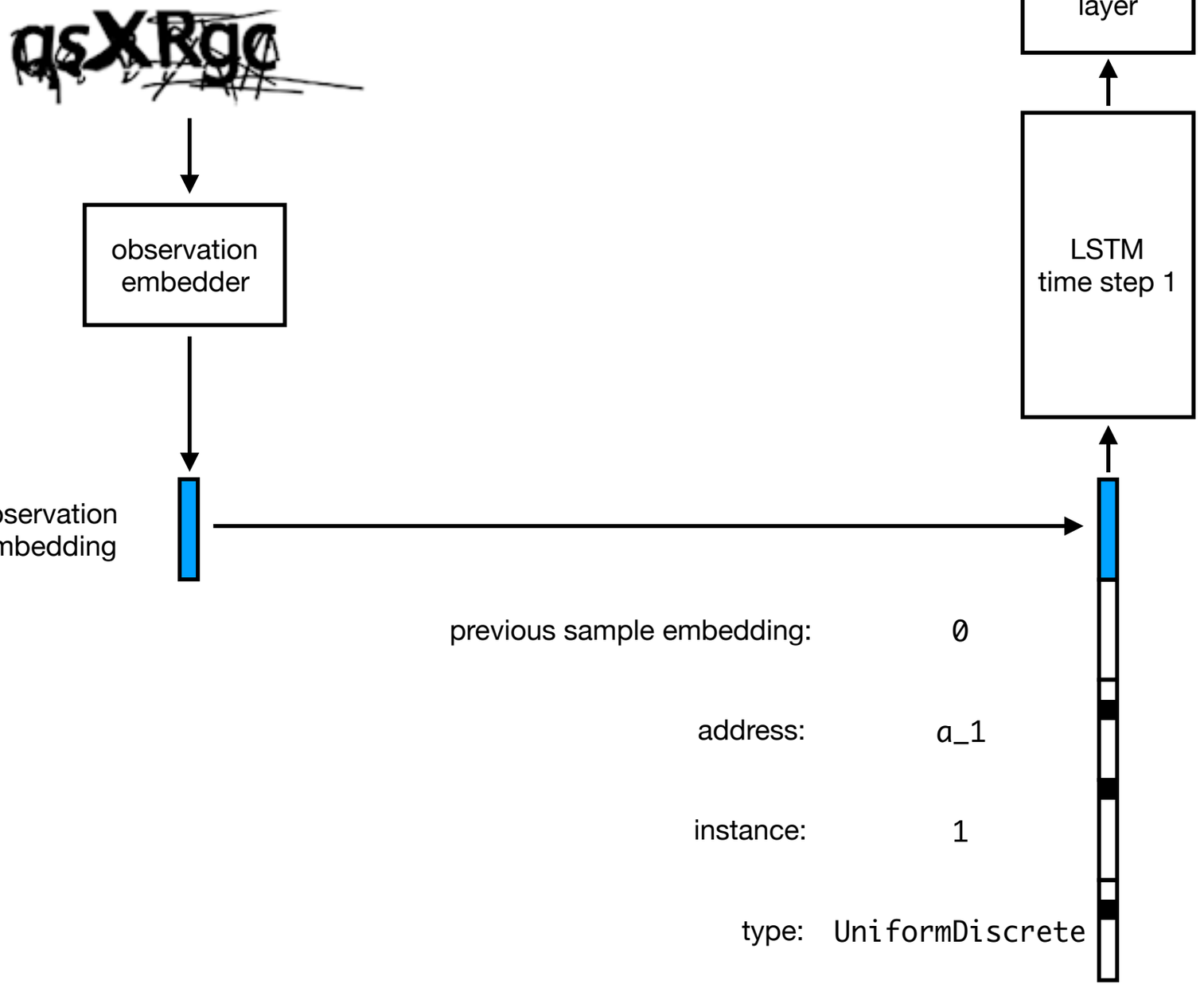
Example: Captcha (Sampling from proposal)

```
letters = []
num_letters = sample(UniformDiscrete(3, 10))
for i in range(num_letters):
   letters.append(sample(Uniform(“a”,…,”z”,“A”,…,”Z”)))

observe(render(letters), observed_captcha)
return letters
```



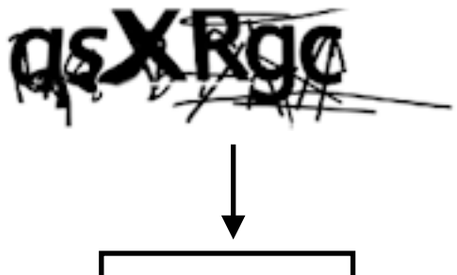
Example: Captcha (Sampling from proposal)

```
letters = []
num_letters = sample(UniformDiscrete(3, 10))
for i in range(num_letters):
   letters.append(sample(Uniform(“a”,…,”z”,“A”,…,”Z”)))

observe(render(letters), observed_captcha)
return letters
```

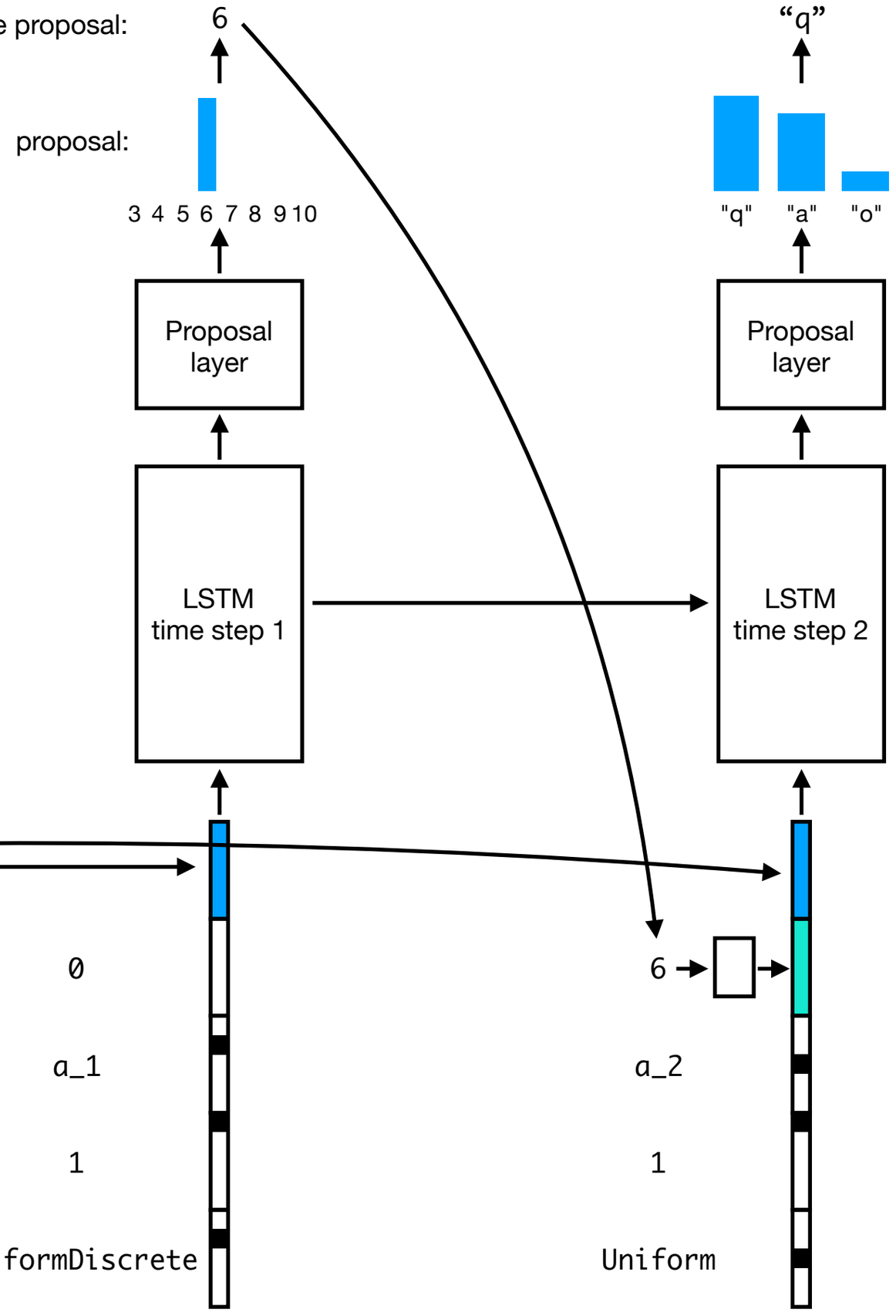


Example: Captcha (Sampling from proposal)

```
letters = []
num_letters = sample(UniformDiscrete(3, 10))
for i in range(num_letters):                          // i = 0
   letters.append(sample(Uniform("a",…,"z","A",…,"Z")))

observe(render(letters), observed_captcha)
return letters
```

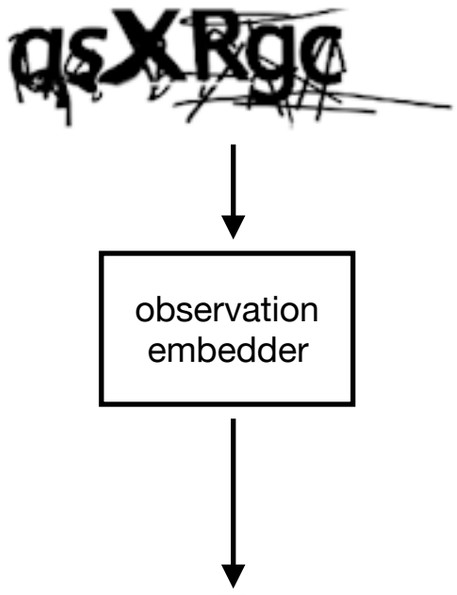


sample from the proposal:



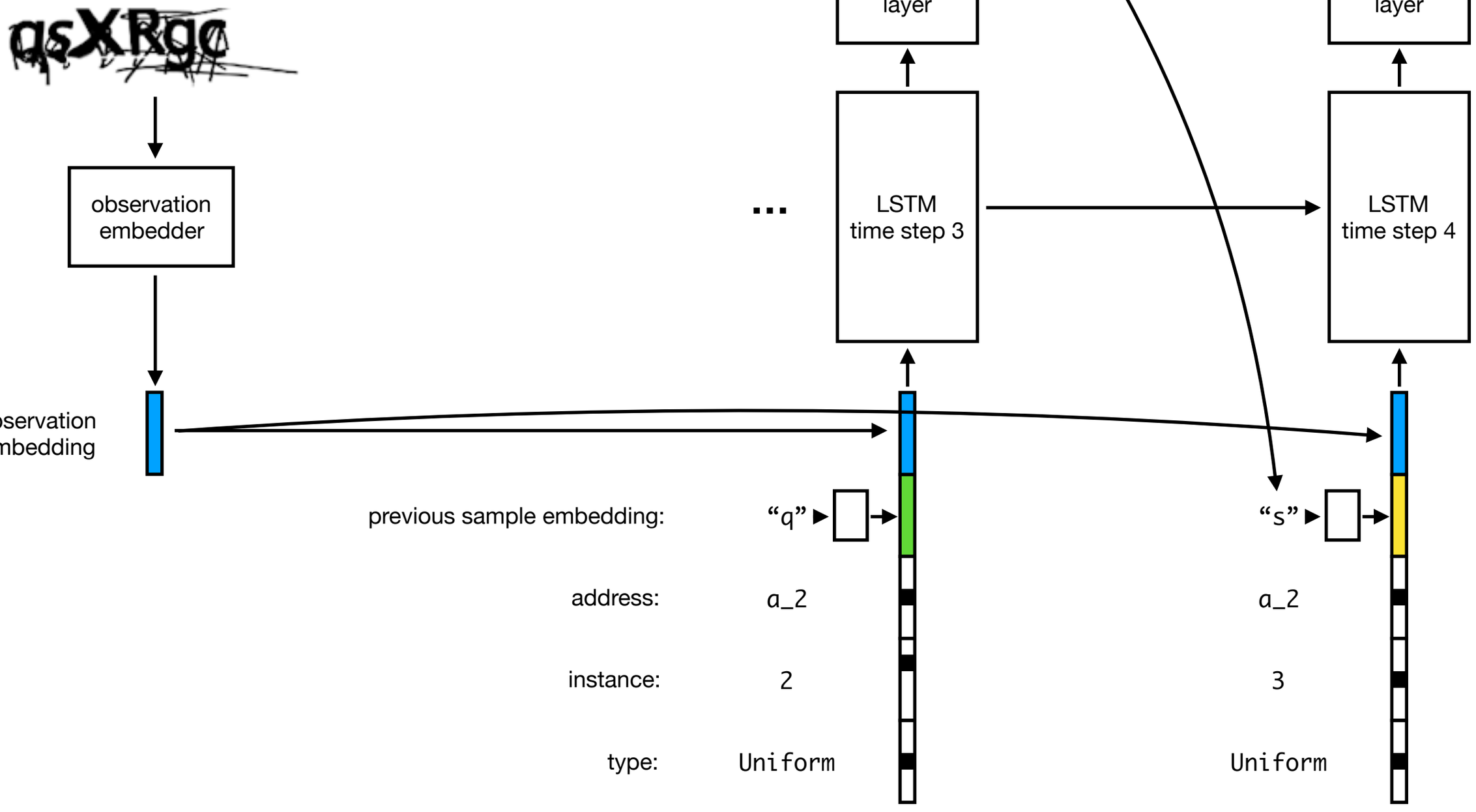
Example: Captcha (Sampling from proposal)

```
letters = []
num_letters = sample(UniformDiscrete(3, 10))
for i in range(num_letters):                          // i = 1
   letters.append(sample(Uniform("a",…,"z","A",…,"Z")))

observe(render(letters), observed_captcha)
return letters
```



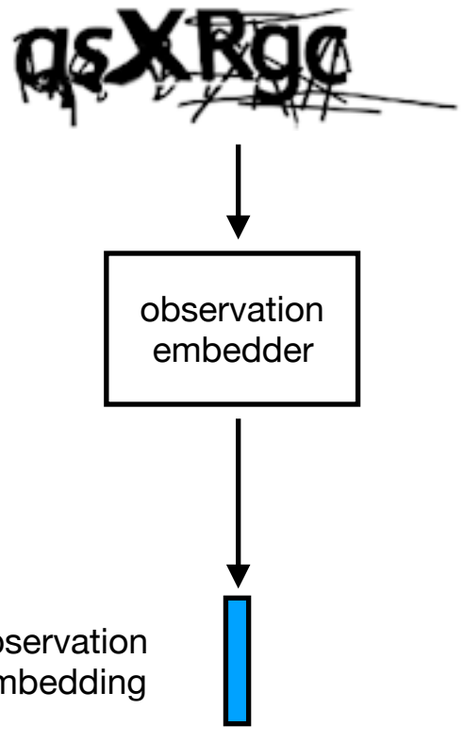
Example: Captcha (Sampling from proposal)

```
letters = []
num_letters = sample(UniformDiscrete(3, 10))
for i in range(num_letters):                          // i = 2
   letters.append(sample(Uniform("a",…,"z","A",…,"Z")))

observe(render(letters), observed_captcha)
return letters
```



Example: Captcha (Sampling from proposal)

```
letters = []
num_letters = sample(UniformDiscrete(3, 10))
for i in range(num_letters):                          // i = …
   letters.append(sample(Uniform(“a”,…,”z”,“A”,…,”Z”)))

observe(render(letters), observed_captcha)
return letters
```

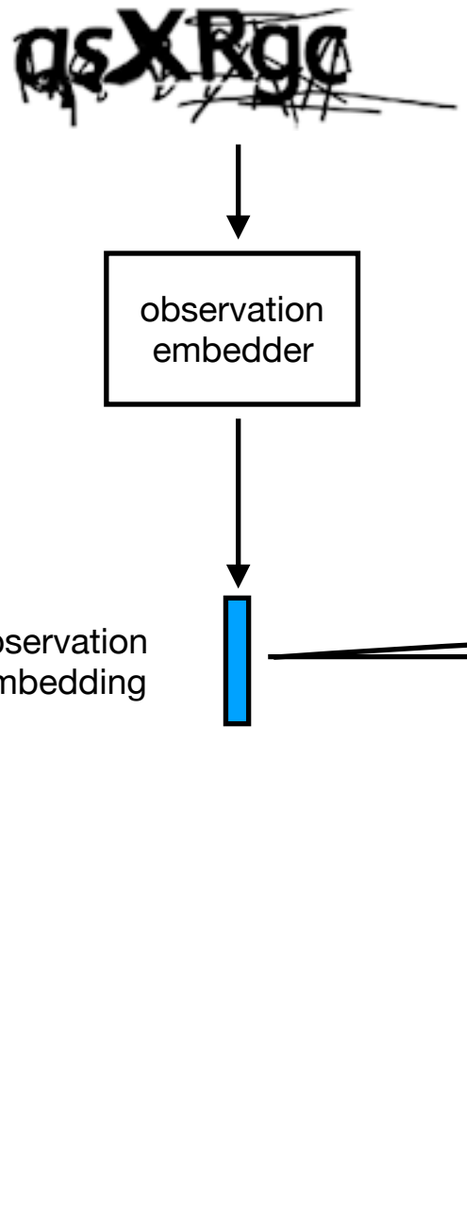


…

etc.

Example: Captcha (Sampling from proposal)

```
letters = []
num_letters = sample(UniformDiscrete(3, 10))
for i in range(num_letters):                          // i = 5
   letters.append(sample(Uniform(“a”,…,”z”,“A”,…,”Z”)))

observe(render(letters), observed_captcha)
return letters
```



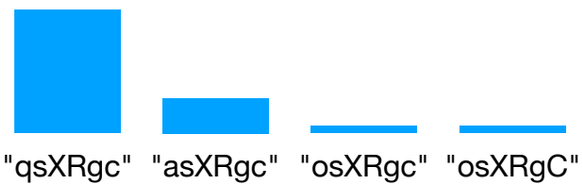
Importance sampling

(6, “q”, “s”, “X”, “R”, “g”, “c”)
Can have different length depending on `num_letters`

For $k = 1, \ldots, K$:

Propose: $\mathbf{x}_k \sim q(\mathbf{x}|\varphi(\eta, \mathbf{y}))$

Weight: $w_k = \frac{p(\mathbf{x}_k, \mathbf{y})}{q(\mathbf{x}_k|\lambda)}$



Normalize: $W_k = \frac{w_k}{\sum_{\ell=1}^{K} w_\ell}$

Approximate:

$$p(\mathbf{x}|\mathbf{y}) \approx \sum_{k=1}^{K} W_k \delta_{\mathbf{x}_k}(\mathbf{x})$$

$$\mathbb{E}_{p(\mathbf{x}|\mathbf{y})}[f(\mathbf{x})] \approx \sum_{k=1}^{K} W_k f(\mathbf{x}_k)$$

$$p(\mathbf{y}) \approx \sum_{k=1}^{K} w_k$$

Le, T. A., Baydin, A. G., & Wood, F. (2017). Inference Compilation and Universal Probabilistic Programming. AISTATS (Vol. 54, pp. 1338–1348).

Compilation: Infinite Training Data

We generate minibatches of program traces

$$\mathcal{D}_{\text{train}} = \left\{ \left(x_t^{(m)}, a_t^{(m)}, i_t^{(m)} \right)_{t=1}^{T^{(m)}}, \left(y_n^{(m)} \right)_{n=1}^{N} \right\}_{m=1}^{M}$$

from the model $p(\mathbf{x}, \mathbf{y})$.

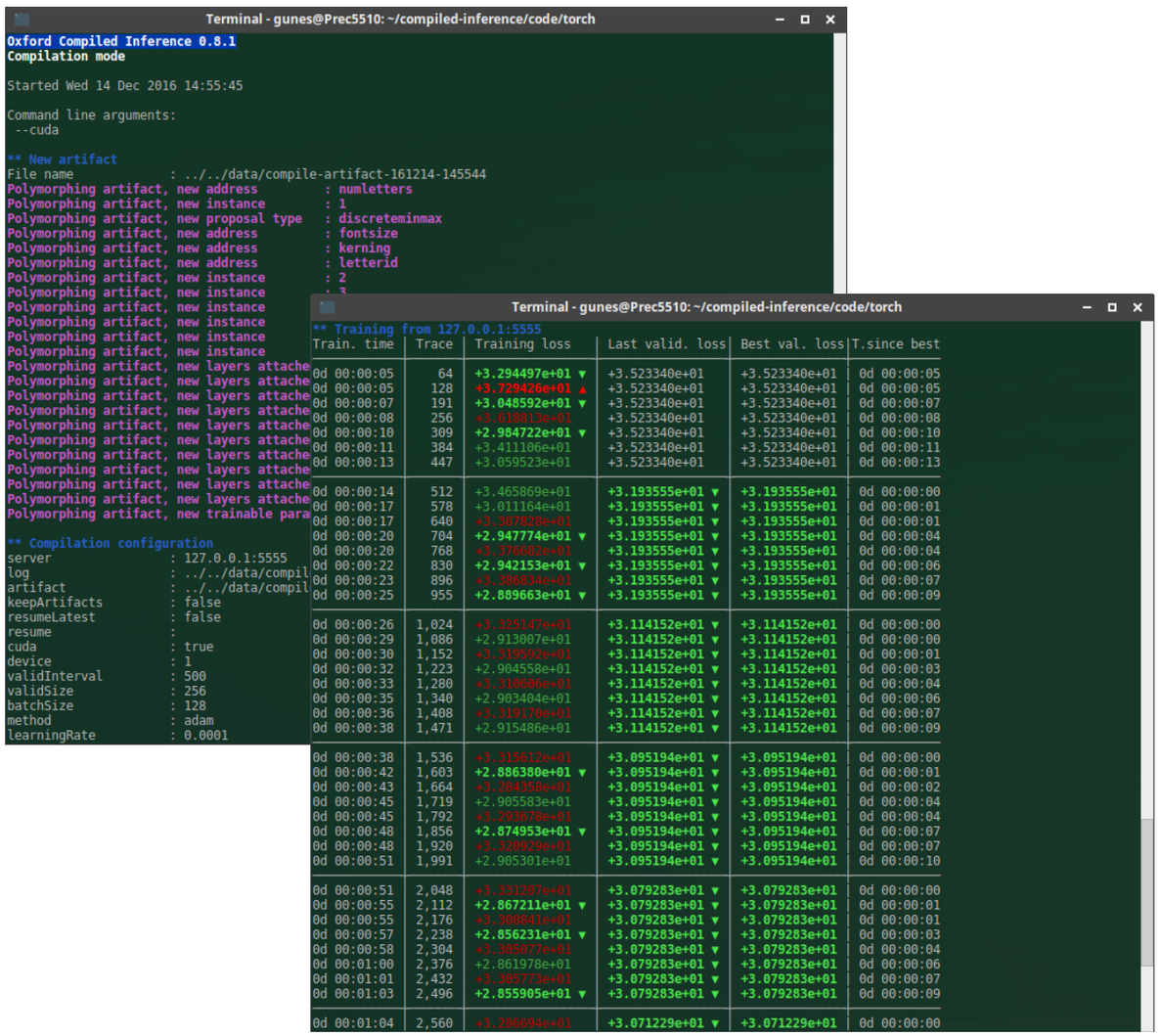
This fixes the **inputs**, **outputs** and **length** of the LSTM.

Train using stochastic gradient descent.

Le, T. A., Baydin, A. G., & Wood, F. (2017). Inference Compilation and Universal Probabilistic Programming. AISTATS (Vol. 54, pp. 1338–1348).

Implementation

```
;; CAPTCHA Query
(with-primitive-procedures [render abc-dist repeatedly]
  (defquery captcha [baseline-image]
    (let [;; Render parameters
          num-letters (sample "numletters" (uniform-discrete 6 8))
          font-size (sample "fontsize" (uniform-discrete 38 44))
          kerning (sample "kerning" (uniform-discrete -2 2))
          letter-ids (repeatedly num-letters #(sample "letterid"
(uniform-discrete 0 (count letter-dict))))
          letters (apply str (map (partial nth letter-dict) letter-ids))

          ;; Render image
          rendered-image (render letters font-size kerning)]

      ;; ABC-style observe
      (observe (abc-dist rendered-image abc-sigma) baseline-image)

      letters)))
```



- Anglican* + PyTorch + ZeroMQ
- Compilation and inference on GPUs
- Models and artifacts can live on separate machines and distributed
- Code & tutorials on GitHub soon

* http://www.robots.ox.ac.uk/~fwood/anglican/

Le, T. A., Baydin, A. G., & Wood, F. (2017). Inference Compilation and Universal Probabilistic Programming. AISTATS (Vol. 54, pp. 1338–1348).

Examples

- Rich, structured, *interpretable* models
  - Open-universe Gaussian Mixture Model
  - Inverse graphics, e.g., Captcha generative model
  - Physics and engineering simulations — ongoing work

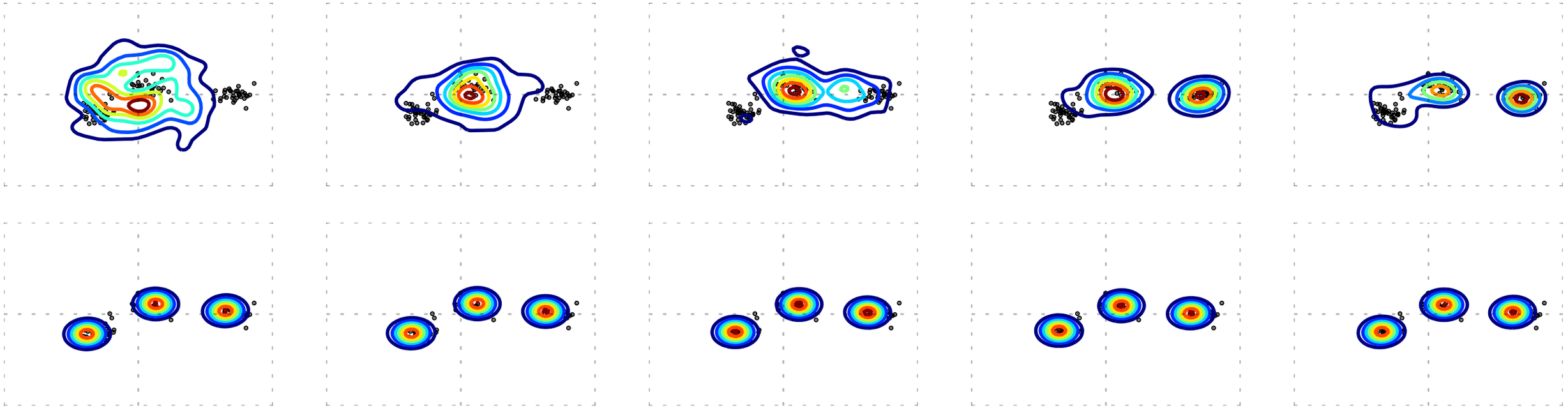
Le, T. A., Baydin, A. G., & Wood, F. (2017). Inference Compilation and Universal Probabilistic Programming. AISTATS (Vol. 54, pp. 1338–1348).

Open-Universe Gaussian Mixture Model

procedure GMM
  $K \sim p(K|\cdot)$ ▷ `sample` number of clusters
  for $k = 1, \ldots, K$ **do**
    $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k \sim p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k|\cdot)$ ▷ `sample` cluster parameters
  $\boldsymbol{\pi} \leftarrow \text{uniform}(1, K)$
  for $n = 1, \ldots, N$ **do**
    $z_n \sim p(z_n|\boldsymbol{\pi})$ ▷ `sample` class label
    $y_n \sim p(y_n|z_n = k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ ▷ `sample` *or* `observe` data
  return $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K, K$

Le, T. A., Baydin, A. G., & Wood, F. (2017). Inference Compilation and Universal Probabilistic Programming. AISTATS (Vol. 54, pp. 1338–1348).

GMM Inference



Particles: 1 10 100 1000 10000

Kernel density estimation of the distribution over maximum a-posteriori values of the means $\{\max_{\mu_k} p(\mu_k|\mathbf{y})\}_{k=1}^{3}$ over 50 independent runs

Top: Sequential Monte Carlo
Bottom: Inference Compilation

Le, T. A., Baydin, A. G., & Wood, F. (2017). Inference Compilation and Universal Probabilistic Programming. AISTATS (Vol. 54, pp. 1338–1348).

Effect of Training Duration

Gaussian Mixtures / Counting & Localization

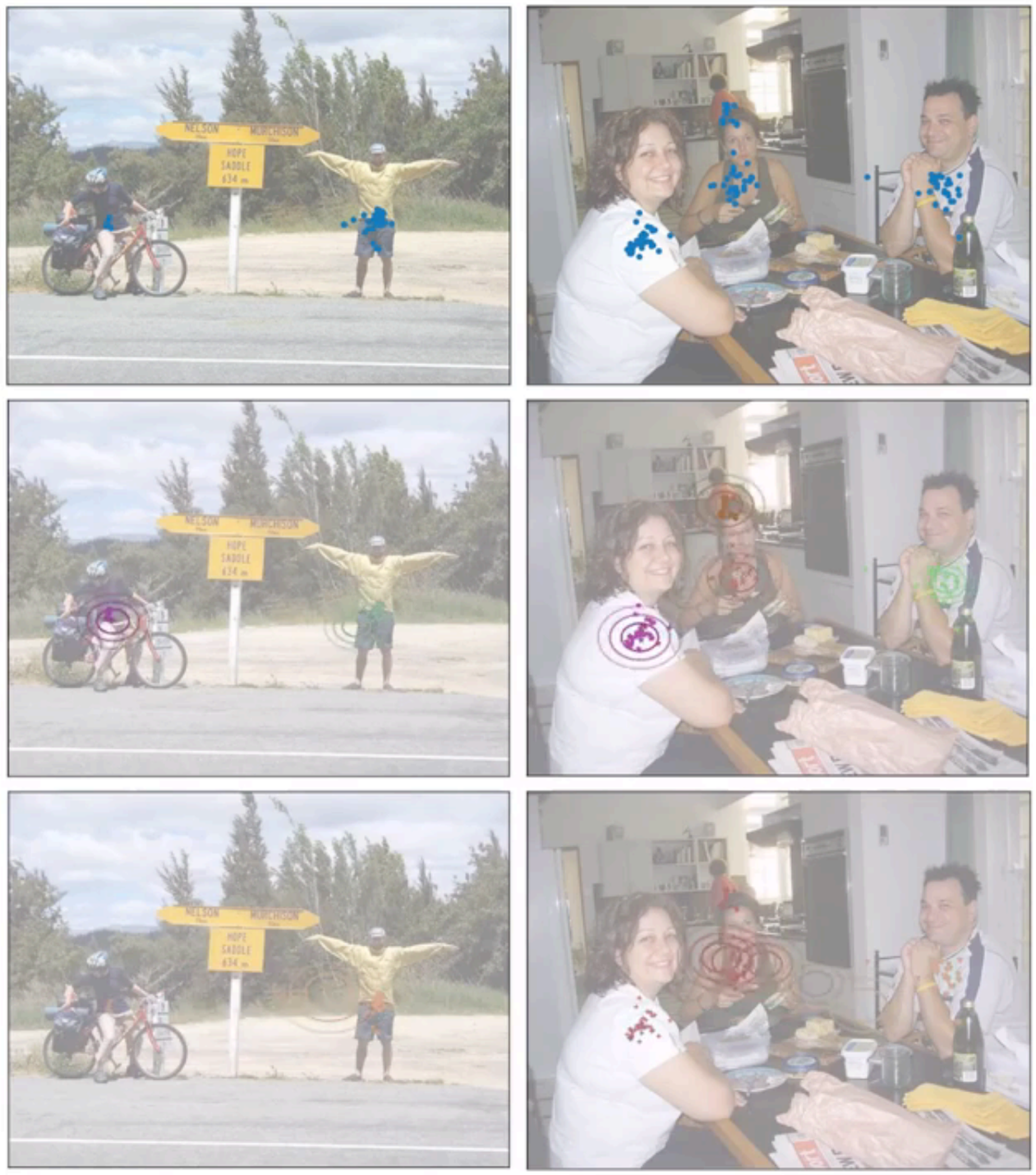
Observed points
(From a Fast R-CNN detector)

Inference

10^7

Training traces

10^4



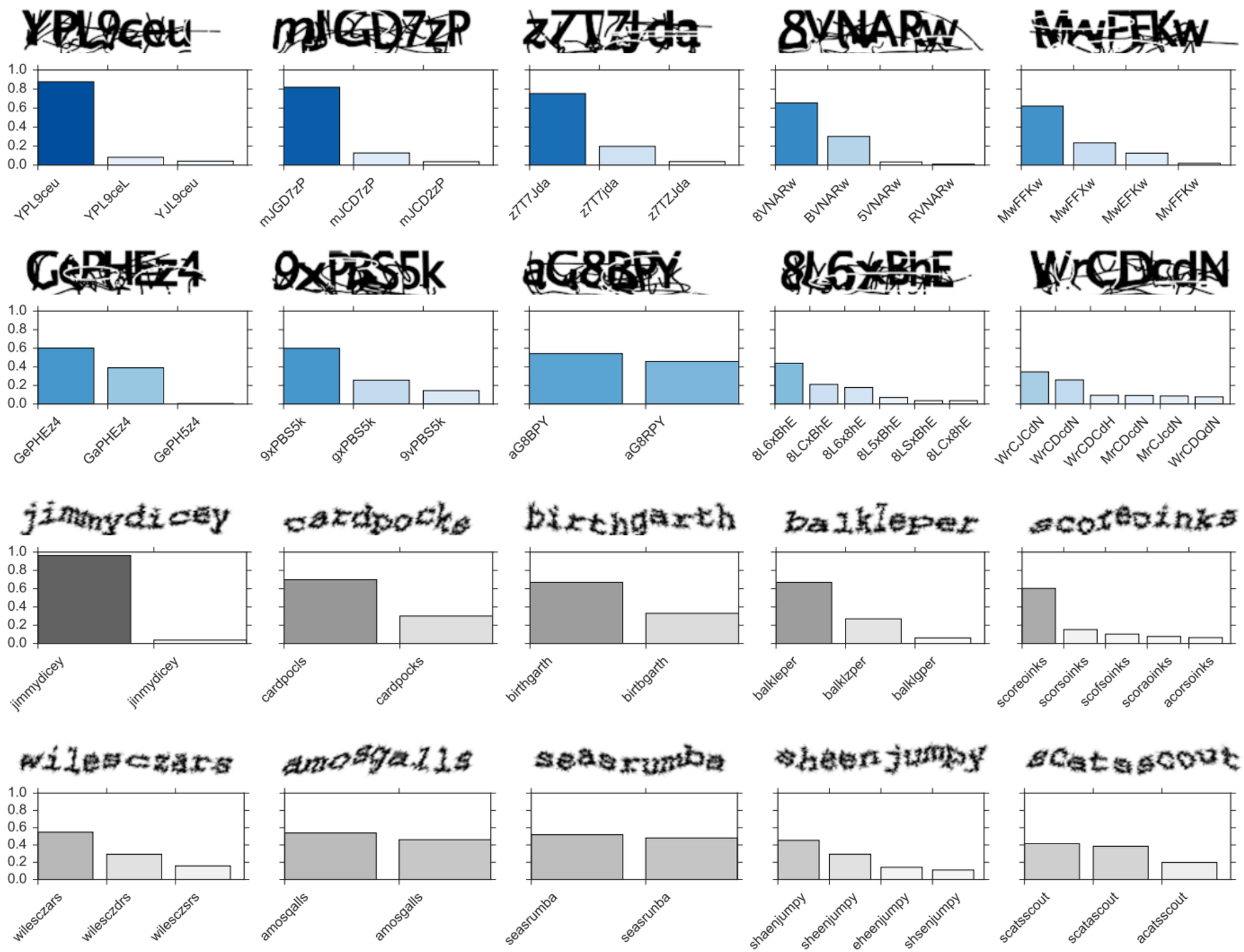
Le, T. A., Baydin, A. G., & Wood, F. (2017). Inference Compilation and Universal Probabilistic Programming. AISTATS (Vol. 54, pp. 1338–1348).

Captcha Generative Model

procedure CAPTCHA
  $\nu \sim p(\nu)$ ▷ **sample** number of letters
  $\kappa \sim p(\kappa)$ ▷ **sample** kerning value
  Generate letters:
  $\Lambda \leftarrow \{\}$
  for $i = 1, \ldots, \nu$ **do**
    $\lambda \sim p(\lambda)$ ▷ **sample** letter ID
    $\Lambda \leftarrow \text{append}(\Lambda, \lambda)$
  Render:
  $\gamma \leftarrow \text{render}(\Lambda, \kappa)$
  $\pi \sim p(\pi)$ ▷ **sample** noise parameters
  $\gamma \leftarrow \text{noise}(\gamma, \pi)$
  return γ

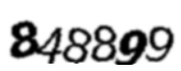
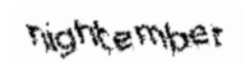
Le, T. A., Baydin, A. G., & Wood, F. (2017). Inference Compilation and Universal Probabilistic Programming. AISTATS (Vol. 54, pp. 1338–1348).

Captcha Breaking Results



Le, Baydin, Zinkov & Wood. Using Synthetic Data to Train Neural Networks is Model-Based Reasoning IJCNN 2017.

Captcha Breaking Results

Type		Baidu (2011) 	Baidu (2013) 	eBay 	Yahoo 	reCaptcha 	Wikipedia 	Facebook 
Our method	RR	99.8%	99.9%	99.2%	98.4%	96.4%	93.6%	91.0%
	BT	72 ms	67 ms	122 ms	106 ms	78 ms	90 ms	90 ms
Bursztein et al. [15]	RR	38.68%	55.22%	51.39%	5.33%	22.67%	28.29%	
	BT	3.94 s	1.9 s	2.31 s	7.95 s	4.59 s		
Starostenko et al. [16]	RR				91.5%	54.6%		
	BT					< 0.5 s		
Gao et al. [17]	RR	34%			55%	34%		
Gao et al. [18]	RR		51%		36%			
	BT		7.58 s		14.72 s			
Goodfellow et al. [6]	RR					99.8%		
Stark et al. [8]	RR					90%		

Facebook Captcha

Observed images

Inference

Training traces				
	10^7	W4kgvQ	uV7EeWB	MqhnpT
	10^6	WA4rjvQ	uV7FeWB	MypppT
	10^5	Woxewd9	mTTEMMm	RIrpES
	10^4	BKvu2Q	C9QDsoN	rS5FP2B

vicarious

$40M raise

Le, Baydin, Zinkov & Wood. Using Synthetic Data to Train Neural Networks is Model-Based Reasoning IJCNN 2017.

Summary

- Probabilistic programming
  - Unified model denotation
  - Automated inference
- Deep learning
  - Powerful regressor
  - Automatic differentiation
  - Needs lots of labeled training data
- Inference compilation
  - Fast repeated inference
  - Training data for deep learning models

Exercise: Write generative model for Captcha in Anglican