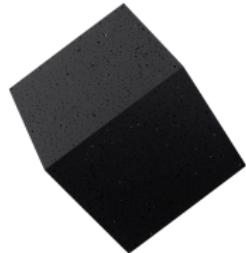


Deep Probabilistic Programming

with Edward

Dustin Tran
Columbia University, OpenAI





Alp Kucukelbir



Adjji Dieng



Dawen Liang



Eugene Brevdo



Maja Rudolph



Matt Hoffman



Rajesh Ranganath



Andrew Gelman



David Blei



Kevin Murphy

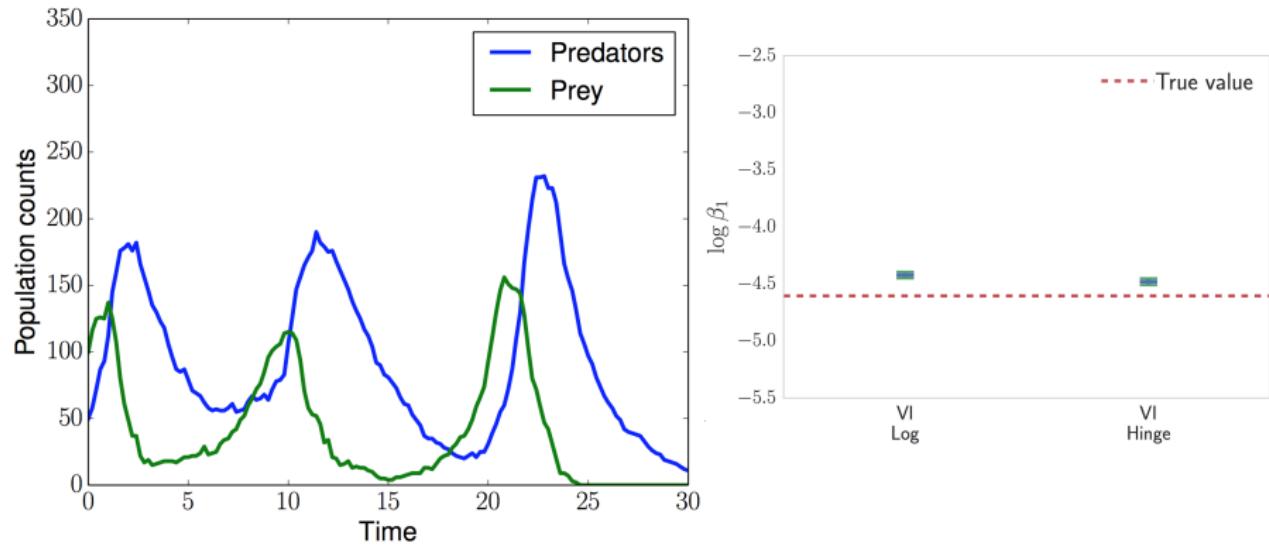


Rif Saurous



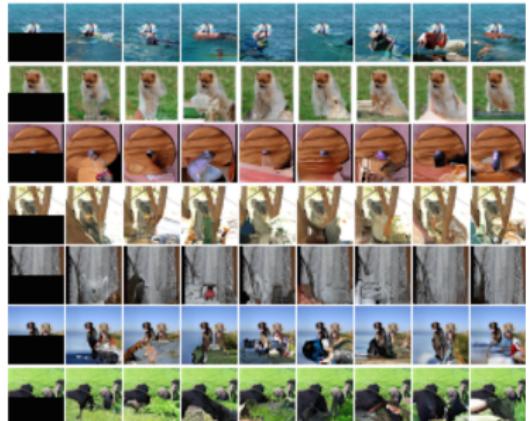
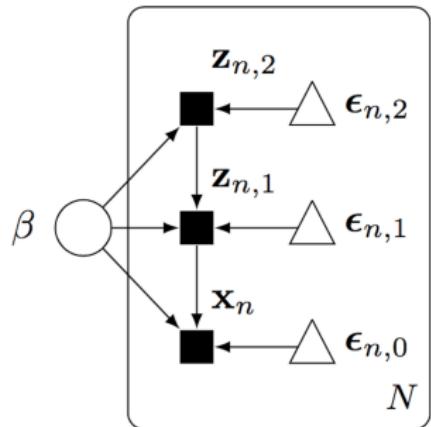
Exploratory analysis of 1.7M taxi trajectories, in Stan

[Kucukelbir+ 2017]



Simulators of 100K time series in ecology, in Edward

[Tran+ 2017]



Generation & compression of 10M colored 32x32 images, in Edward

[Tran+ 2017; fig from Van der Oord+ 2016]

A solid , spooky entertainment worthy of the price of a ticket .
Of all the Halloween 's , this is the most visually unappealing .

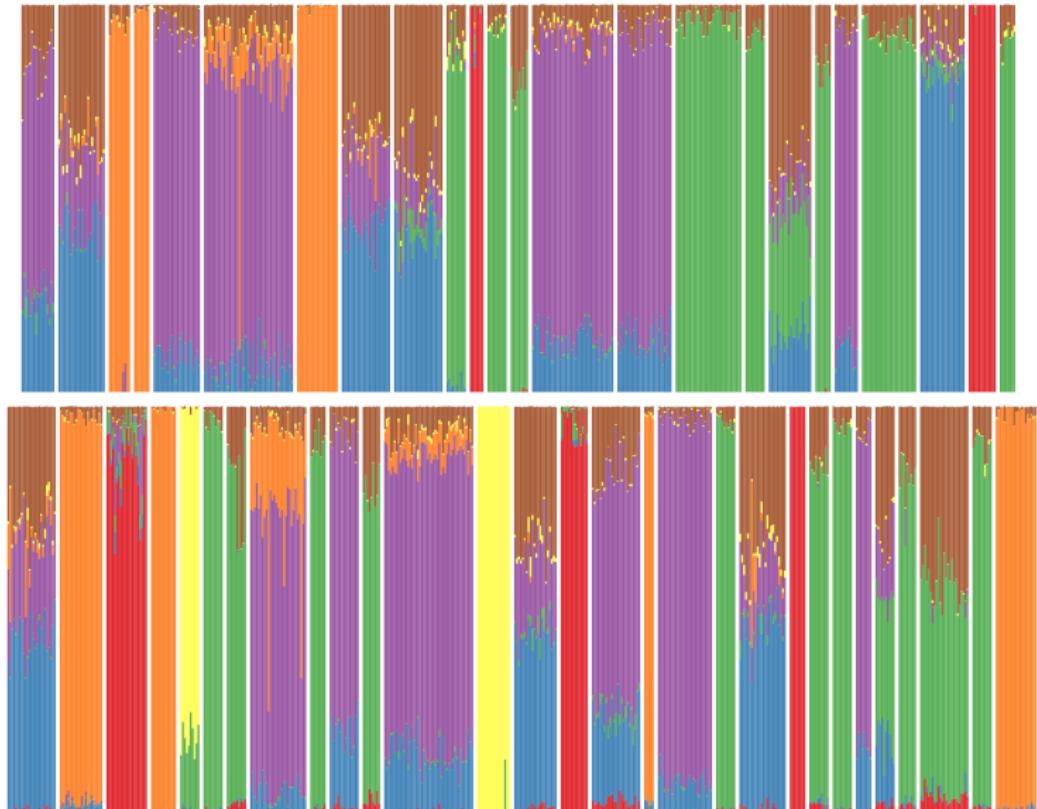
Poetry in motion captured on film .
An imponderably stilted and self-consciously arty movie .

Steers refreshingly clear of the usual cliches .
A profoundly stupid affair , populating its hackneyed and meanspirited
storyline with cardboard characters and performers who value cash above
credibility .

A visual spectacle full of stunning images and effects .
An annoying orgy of excess and exploitation that has no point and
goes nowhere .

Sentiment transfer of unaligned pairs of 8.5K sentences, in Edward

[in preparation]



Cause and effect of 1.6B genetic measurements, in Edward

[in preparation; fig from Gopalan+ 2017]

What is probabilistic programming?

Probabilistic programs reify models from mathematics to physical objects.

- Each model is equipped with memory (“bits”, floating point, storage) and computation (“flops”, scalability, communication).

Anything you do lives in the world of probabilistic programming.

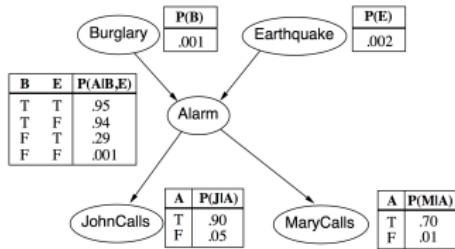
- Any computable model.
 - ex. graphical models; neural networks; SVMs; stochastic processes.
- Any computable inference algorithm.
 - ex. automated inference; model-specific algorithms; inference within inference (learning to learn).
- Any computable application.
 - ex. exploratory analysis; object recognition; code generation; causality.

Simulation hypothesis.

“The universe is a simulation from a computer program.”

(Zuse, Bostrom, Schmidhuber, Musk)

My One Slide on Robotics



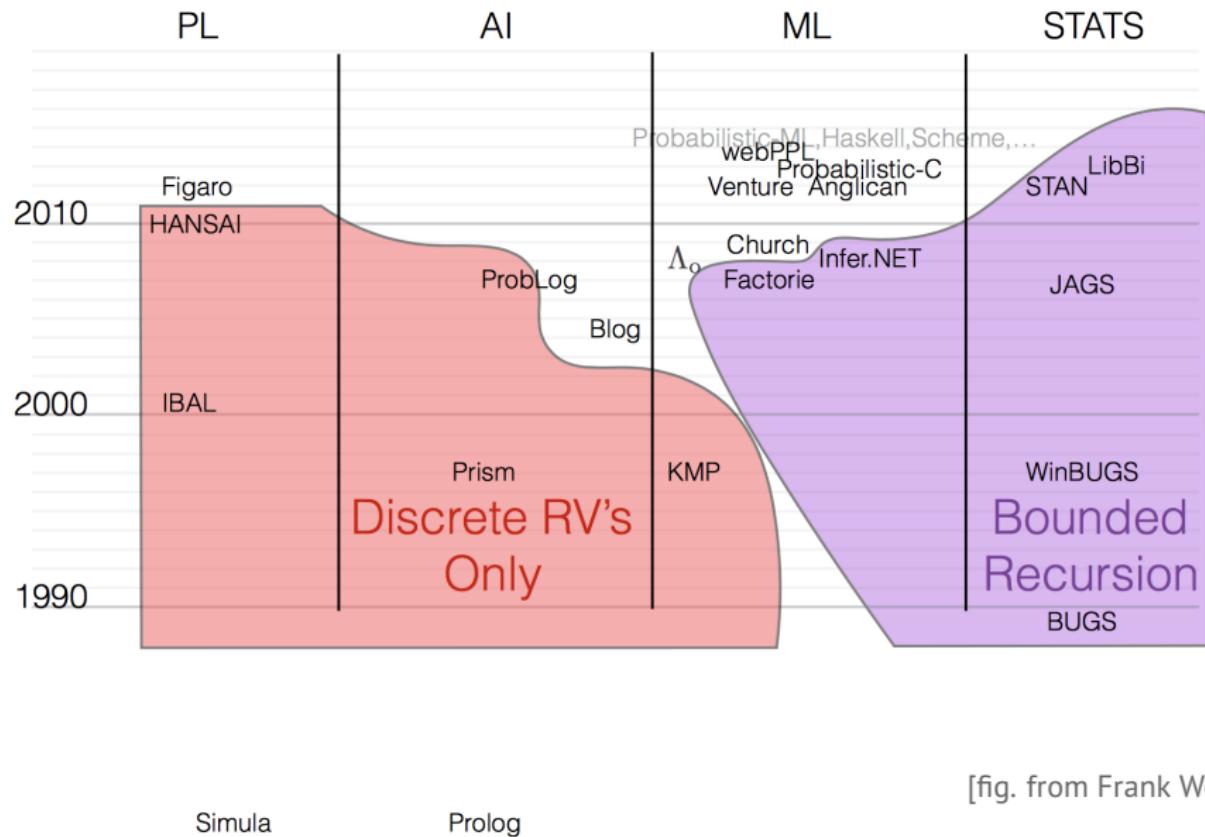
```
1: bool earthquake, burglary, alarm, phoneWorking,  
   maryWakes,called;  
2: earthquake = Bernoulli(0.001);  
3: burglary = Bernoulli(0.01);  
4: alarm = earthquake || burglary;  
5: if(earthquake)  
6:   phoneWorking = Bernoulli(0.6);  
7: else  
8:   phoneWorking = Bernoulli(0.99);  
9: if (alarm && earthquake)  
10:   maryWakes = Bernoulli(0.8);  
11: else if (alarm)  
12:   maryWakes = Bernoulli(0.6);  
13: else  
14:   maryWakes = Bernoulli(0.2);  
15: called = maryWakes && phoneWorking;  
16: observe(called);  
17: return burglary;
```

Bayesian networks have classic roots in cognition and robotics (planning, reasoning, manipulation). (Pearl, 1986)

Probabilistic programming can be thought of as expanding Bayesian networks with programming semantics (control flow, recursion).
(Koller+, 1997; Pfeffer, 2001; Milch+, 2005)

[fig from Gordon+ 2014]

Languages and Systems



George E.P. Box (1919 - 2013)

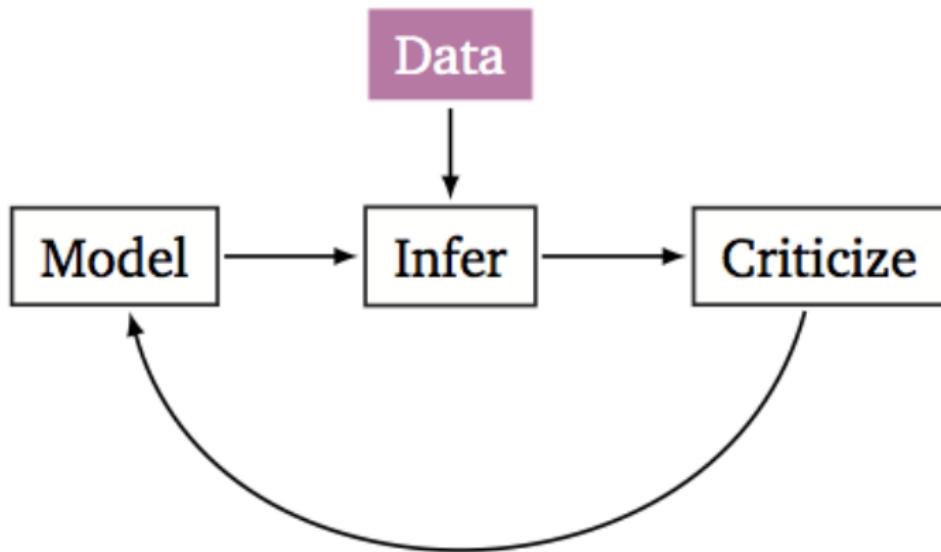


An iterative process for science:

1. Build a model of the science
2. Infer the model given data
3. Criticize the model given data

[Box & Hunter 1962, 1965; Box & Hill 1967; Box 1976, 1980]

Box's Loop



Edward is a library designed around this loop.

[Box 1976, 1980; Blei 2014]

[Code](#)[Issues 110](#)[Pull requests 18](#)[Insights](#)

A library for probabilistic modeling, inference, and criticism. Deep generative models, variational inference. Runs on TensorFlow. <http://edwardlib.org>

[bayesian-methods](#)[deep-learning](#)[machine-learning](#)[data-science](#)[tensorflow](#)[neural-networks](#)[statistics](#)[probabilistic-programming](#)[1,753 commits](#)[14 branches](#)[27 releases](#)[62 contributors](#)Branch: [master](#) ▾[New pull request](#)[Find file](#)[Clone or download](#) ▾

dustinvtran version 1.3.4

Latest commit 2402498 12 hours ago

docker

Use Observations and remove explicit storage of data files (#751)

20 days ago

[Sign Up](#)[Log In](#)[all categories](#) ▾[Latest](#)[Top](#)

Topic

Category

Users

Replies

Views

Activity

Iterative estimators ("bayes filters") in Edward?



5

21

7h

Tutorial for multiple variational methods using Poisson regression?



2

20

1d

blei-lab/edward A library for probabilistic modeling, inference, and criticism. <http://edwardlib.org>

Faiez Shakil @faezs Jan 23 02:47 Hi @dustinvtran, thanks for edward, the library and surrounding literature have been immense fun to get into. Would you be able to tell me whether it'd be relatively painless to get the inference compute graphs from Ed as native tensorflow graphdefs and use them on mobile platforms? Or would I have to port a bunch of custom ops

PEOPLE REPO INFO

We have an active community of several thousand users & many contributors.

How do we use Edward?

Model

A random variable \mathbf{x} is an object parameterized by tensors (multi-dimensional arrays) θ^* .

```
1 # univariate normal
2 Normal(loc=0.0, scale=1.0)
3 # vector of 5 univariate normals
4 Normal(loc=tf.zeros(5), scale=tf.ones(5))
5 # 2 x 3 matrix of Exponentials
6 Exponential(rate=tf.ones([2, 3]))
```

It is equipped with methods such as `log_prob()` and `sample()`.

Each random variable is associated to a tensor \mathbf{x}^* , $\mathbf{x}^* \sim p(\mathbf{x} | \theta^*)$.

Mutable states let random variables condition on values that change, e.g., discriminative models $p(\mathbf{y} | \mathbf{x})$ and model parameters $p(\mathbf{x}; \theta)$.

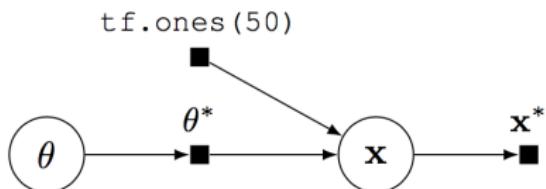
Example: Beta-Bernoulli

Consider a Beta-Bernoulli model,

$$p(\mathbf{x}, \theta) = \text{Beta}(\theta | 1, 1) \prod_{n=1}^{50} \text{Bernoulli}(x_n | \theta),$$

where θ is a probability shared across 50 data points $\mathbf{x} \in \{0, 1\}^{50}$.

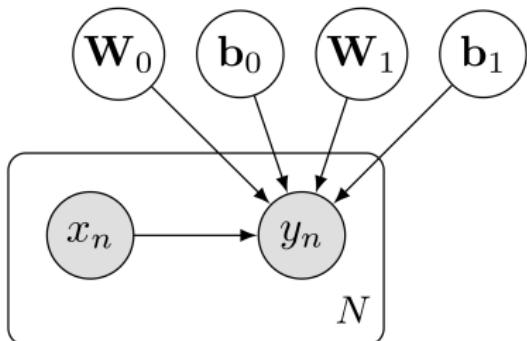
```
1 theta = Beta(1.0, 1.0)
2 x = Bernoulli(probs=tf.ones(50) * theta)
```



Fetching \mathbf{x} from the graph generates a binary vector of 50 elements.

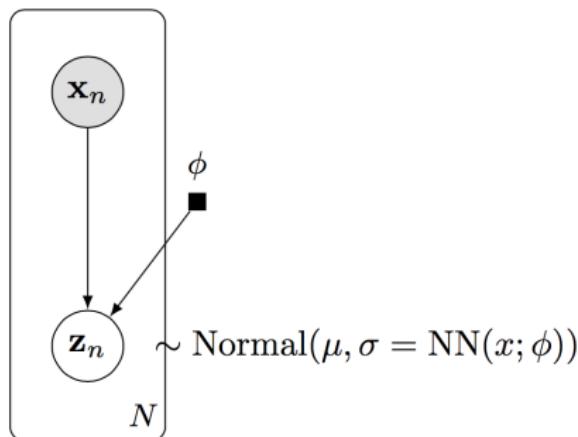
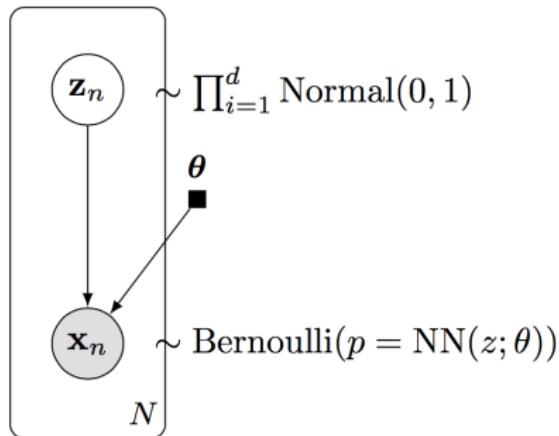
All computation is represented on the graph, enabling us to leverage model structure during inference.

Example: Bayesian neural network for classification



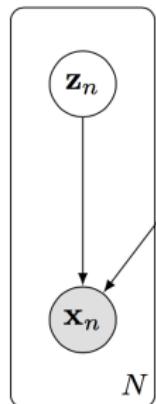
```
1 W_0 = Normal(mu=tf.zeros([D, H]), sigma=tf.ones([D, H]))
2 W_1 = Normal(mu=tf.zeros([H, 1]), sigma=tf.ones([H, 1]))
3 b_0 = Normal(mu=tf.zeros(H), sigma=tf.ones(L))
4 b_1 = Normal(mu=tf.zeros(1), sigma=tf.ones(1))
5
6 x = tf.placeholder(tf.float32, [N, D])
7 y = Bernoulli(logits=tf.matmul(tf.nn.tanh(tf.matmul(x, W_0) + b_0), W_1) + b_1)
```

Example: Variational Auto-Encoder for Binarized MNIST

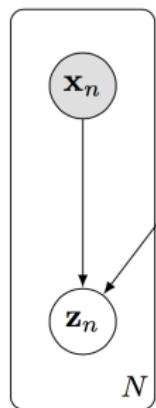


[Kingma & Welling 2014; Rezende+ 2014]

Example: Variational Auto-Encoder for Binarized MNIST



```
# Probabilistic model  
z = Normal(loc=tf.zeros([N, d]), scale=tf.ones([N, d]))  
h = Dense(256, activation='relu')(z)  
x = Bernoulli(logits=Dense(28 * 28, activation=None)(h))
```



```
# Variational model  
qx = tf.placeholder(tf.float32, [N, 28 * 28])  
qh = Dense(256, activation='relu')(qx)  
qz = Normal(loc=Dense(d, activation=None)(qh),  
            scale=Dense(d, activation='softplus')(qh))
```

Inference

Given

- Data $\mathbf{x}_{\text{train}}$.
- Model $p(\mathbf{x}, \mathbf{z}, \boldsymbol{\beta})$ of observed variables \mathbf{x} and latent variables $\mathbf{z}, \boldsymbol{\beta}$.

Goal

- Calculate posterior distribution

$$p(\mathbf{z}, \boldsymbol{\beta} \mid \mathbf{x}_{\text{train}}) = \frac{p(\mathbf{x}_{\text{train}}, \mathbf{z}, \boldsymbol{\beta})}{\int p(\mathbf{x}_{\text{train}}, \mathbf{z}, \boldsymbol{\beta}) d\mathbf{z} d\boldsymbol{\beta}}.$$

This is the key problem in Bayesian inference.

Inference

In Edward, all inference has the same structure.

```
inference = ed.Inference({beta: qbeta, z: qz}, data={x: x_train})
```

Inference has two inputs:

1. latent variables \mathbf{z}, β , binding model variables to approximate factors;
2. observed variables \mathbf{x} , binding model variables to data.

Inference has class methods:

- `run()` runs the algorithm from initialization to convergence;
- `initialize()`, `update()`, `print_progress()`, etc. provides finer control.

Example: Variational Auto-Encoder for Binarized MNIST

```
1 N = 1000 # number of data points
2 d = 10 # latent dimensionality
3
4 # DATA
5 x_data = np.loadtxt('mnist.txt', np.float32)
6
7 # MODEL
8 z = Normal(loc=tf.zeros([N, d]), scale=tf.ones([N, d]))
9 h = Dense(256, activation='relu')(z)
10 x = Bernoulli(logits=Dense(28 * 28, activation=None)(h))
11
12 # INFERENCE
13 qx = tf.placeholder(tf.float32, [N, 28 * 28])
14 qh = Dense(256, activation='relu')(qx)
15 qz = Normal(loc=Dense(d, activation=None)(qh),
16               scale=Dense(d, activation='softplus')(qh))
17
18 inference = ed.KLqp({z: qz}, data={x: x_data, qx: qx})
19 inference.run()
```

Example: Variational Auto-Encoder for Binarized MNIST

```
1 N = 1000 # number of data points
2 d = 10 # latent dimensionality
3
4 # DATA
5 x_data = np.loadtxt('mnist.txt', np.float32)
6
7 # MODEL
8 z = Normal(loc=tf.zeros([N, d]), scale=tf.ones([N, d]))
9 h = Dense(256, activation='relu')(z)
10 x = Bernoulli(logits=Dense(28 * 28, activation=None)(h))
11
12 # INFERENCE
13 T = 10000 # number of samples
14 qz = Empirical(params=tf.Variable(tf.zeros([T, N, d])))
15
16 inference = ed.HMC({z: qz}, data={x: x_data})
17 inference.run()
```

Non-Bayesian Methods: GANs

GANs posit a generative process,

$$\begin{aligned}\epsilon &\sim \text{Normal}(0, 1) \\ \mathbf{x} &= G(\epsilon; \theta)\end{aligned}$$

for some generative network G .

Training uses a discriminative network D via the optimization problem

$$\min_{\theta} \max_{\phi} \mathbb{E}_{p^*(\mathbf{x})} [\log D(\mathbf{x}; \phi)] + \mathbb{E}_{p(\mathbf{x}; \theta)} [\log(1 - D(\mathbf{x}; \phi))]$$

The generator tries to generate samples indistinguishable from true data.

The discriminator tries to discriminate samples from the generator and samples from the true data.

Non-Bayesian Methods: GANs

```
1 def generative_network(eps):
2     h = Dense(256, activation='relu') (eps)
3     return Dense(28 * 28, activation=None) (h)
4
5 def discriminative_network(x):
6     h = Dense(28 * 28, activation='relu') (x)
7     return Dense(h, activation=None) (1)
8
9 # Probabilistic model
10 eps = Normal(loc=tf.zeros([N, d]), scale=tf.ones([N, d]))
11 x = generative_network(eps)
12
13 inference = ed.GANInference(data={x: x_train},
14                               discriminator=discriminative_network)
15 inference.run()
```

Non-Bayesian Methods: GANs

```
1 def generative_network(eps) :
2     h = Dense(256, activation='relu') (eps)
3     return Dense(28 * 28, activation=None) (h)
4
5 def discriminative_network(x) :
6     h = Dense(28 * 28, activation='relu') (x)
7     return Dense(h, activation=None) (1)
8
9 # Probabilistic model
10 eps = Normal(loc=tf.zeros([N, d]), scale=tf.ones([N, d]))
11 x = generative_network(eps)
12
13 inference = ed.WGANInference(data={x: x_train},
14                                discriminator=discriminative_network)
15 inference.run()
```

Experiment: GPU-accelerated Hamiltonian Monte Carlo

Probabilistic programming system	Runtime (s)
Handwritten NumPy (1 CPU)	534
Stan (1 CPU) [Carpenter+ 2016]	171
PyMC3 (12 CPU) [Salvatier+ 2015]	30.0
Edward (12 CPU)	8.2
Handwritten TensorFlow (GPU)	5.0
Edward (GPU)	4.9 (35x faster than Stan)

Run HMC for 100 iterations and fixed hyperparameters.

Bayesian logistic regression for Covertype (581012 data points, 54 features).

12-core Intel i7-5930K CPU at 3.50GHz and NVIDIA Titan X (Maxwell) GPU.
Single precision.

Edward is orders of magnitude faster than existing software for large data.

[Tran+ 2017]

Experiment: Neural Machine Translation (EN-DE)

Method	# of Supervised Examples (BLEU)		
	500K	1M	2M
Transformer (65M params)	12.3	14.6	21.3
Transformer (130M params)	10.3	19.5	22.7
Feature matching (130M params)	18.5	22.4	24.0

WMT 2014 English-to-German (\approx 4.5M sentences, 222M tokens). Training set is N parallel sentences, rest are non-parallel.

Test on newstest2014; validation on newstest2013.

SOTA is [Vaswani+ 2017] (28.4); Conv Seq2Seq (25.16).

SOTA 2016 is ByteNet (23.75); Google NMT (24.6).

SOTA 2015 is RNN Enc-Dec-Att (20.9); RNN Enc-Dec (14.0).

Summary

1. Edward is a probabilistic programming system designed for fast experimentation.

It emphasizes fine tuning of inference alongside model development.

2. Edward is integrated into TensorFlow.

It features significant speedups over existing systems on large data or multi-device / multi-machine environments.

Current directions

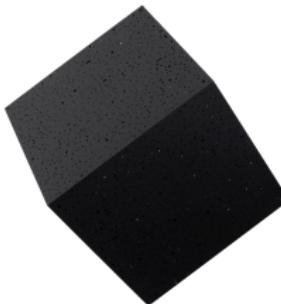
We are extending Edward with new semantics.

1. Dynamic computational graphs. (Google)
2. Benchmarks across modeling & inference applications.
3. A one-line API for loading standard data sets in machine learning.

We are applying Edward for AI and scientific research.

1. Causal models for genome wide association studies. (Blei)
2. Alignment & data efficiency in language. (OpenAI)
3. AGI (Solomonoff induction) by learning probabilistic programs. (OpenAI)

References



edwardlib.org

Two NIPS workshops this year:

- Approximate Bayesian Inference (approximateinference.org).
- Bayesian Deep Learning (bayesiandeeplearning.org).

Two NIPS papers this year:

- **D. Tran**, R. Ranganath, and D.M. Blei.
Deep and hierarchical implicit models.
- A. Dieng, **D. Tran**, R. Ranganath, and D.M. Blei.
The χ -divergence for approximate inference.