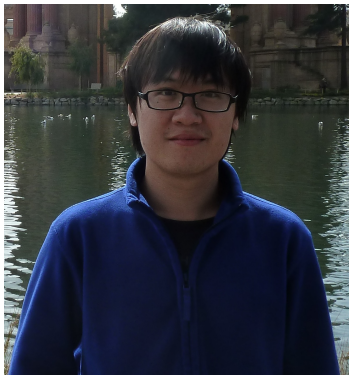
Deep Probabilistic Programming

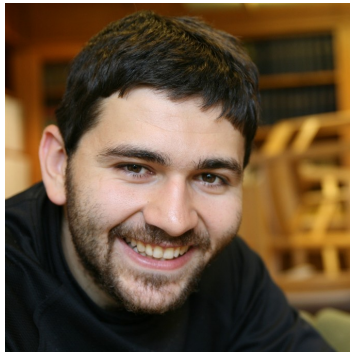
Dustin Tran
Columbia University, OpenAI

Alp Kucukelbir

Adji Dieng



Dawen Liang



Eugene Brevdo

Maja Rudolph

Matt Hoffman

Rajesh Ranganath

Andrew Gelman

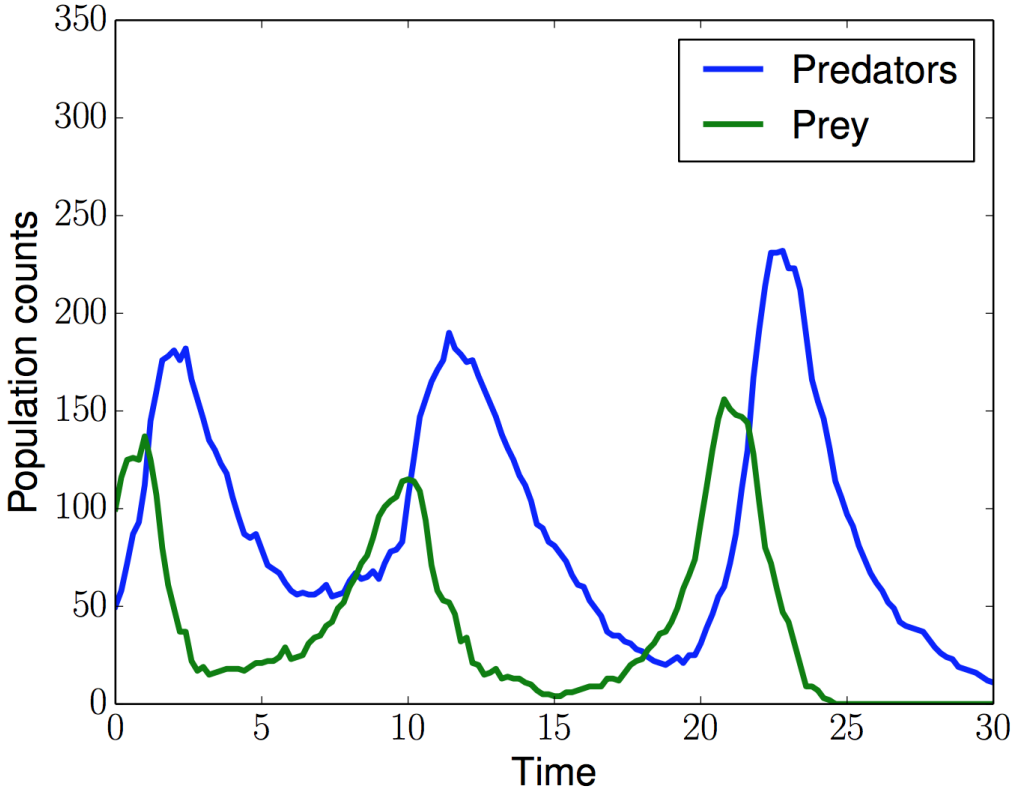
David Blei

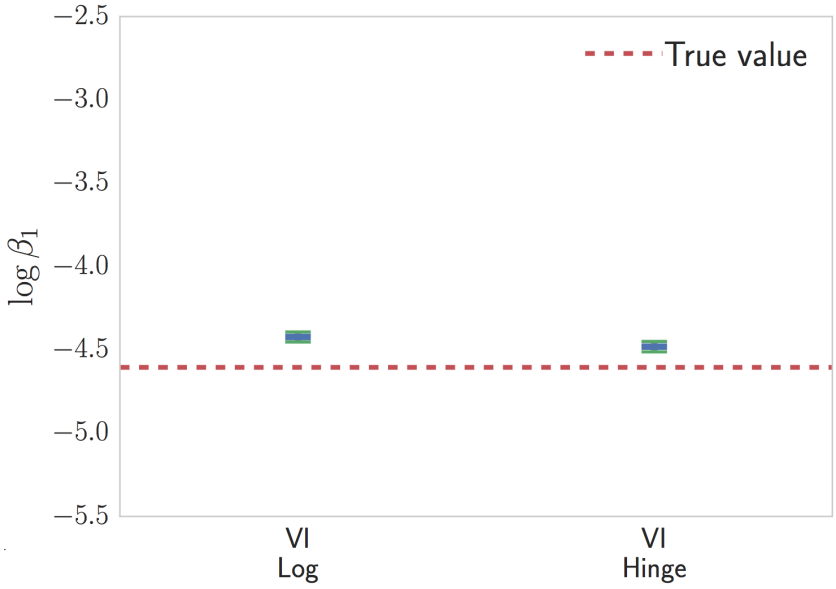
Kevin Murphy

Rif Saurous

Exploratory analysis of 1.7M taxi trajectories, in Stan

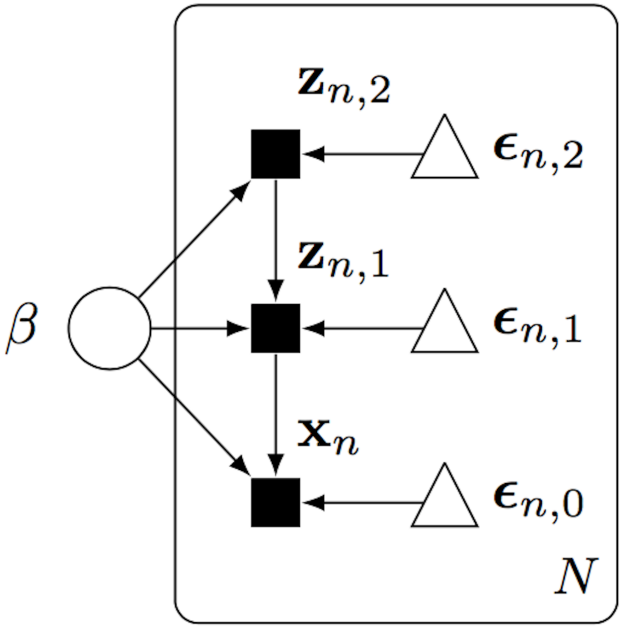
[Kucukelbir+ 2017]

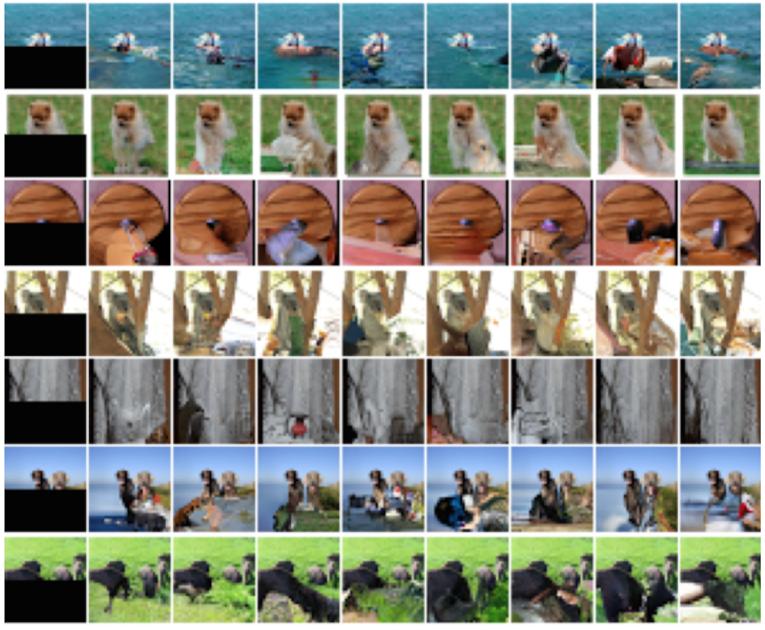




Simulators of 100K time series in ecology, in Edward

[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 .

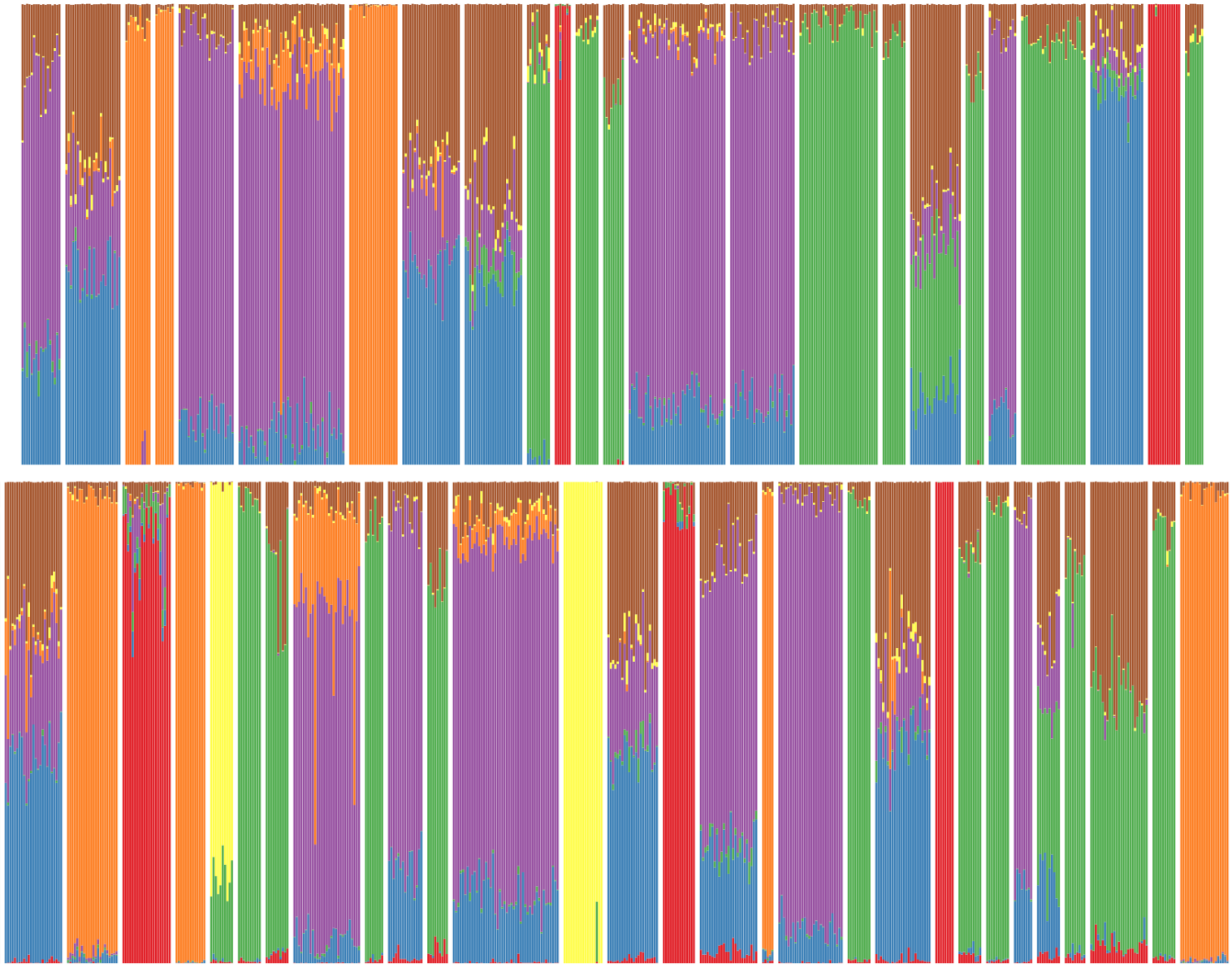
Poetry in motion captured on film .
An imponderably stilted and self-consciously arty movie .

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.

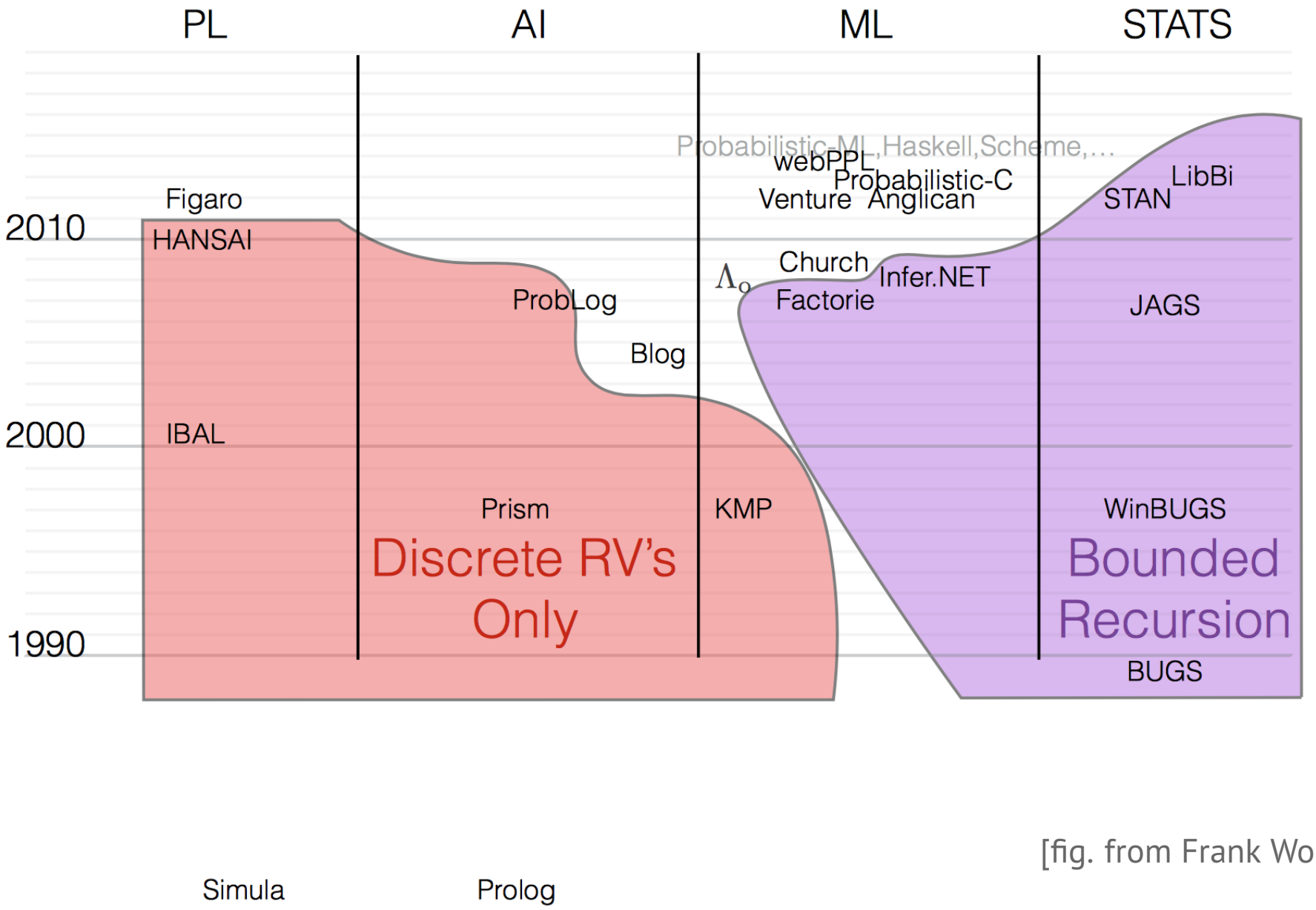
  ex. exploratory analysis; object recognition; code generation; causality.

Simulation hypothesis.

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

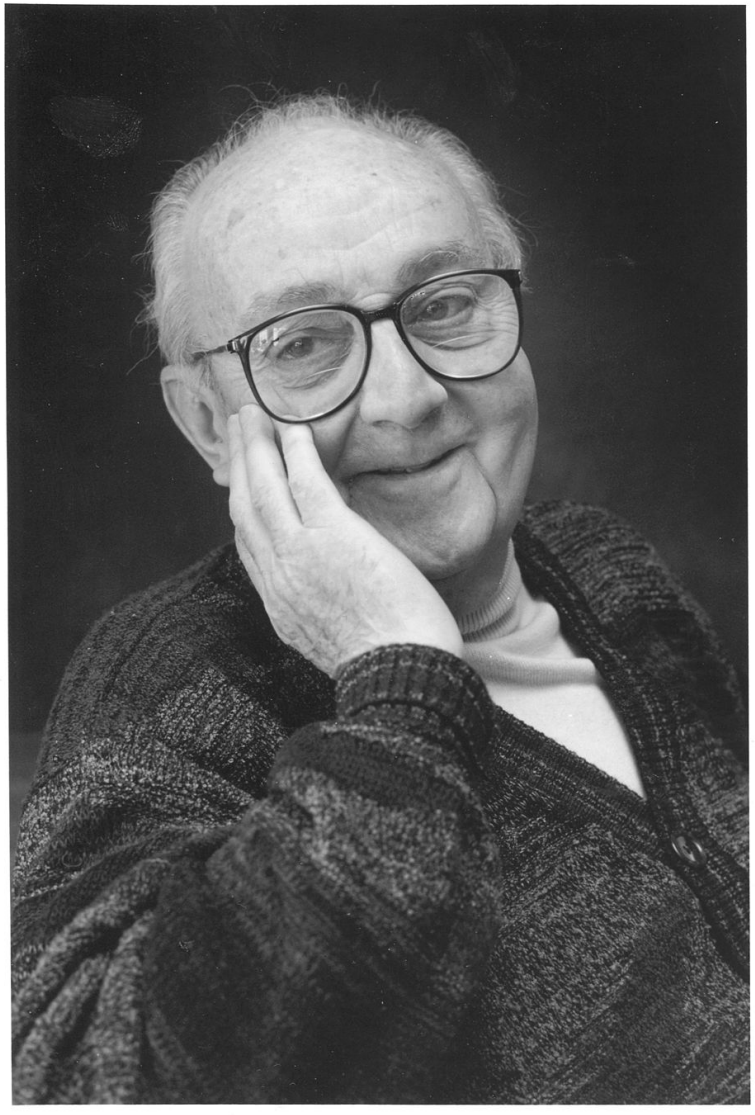
(Zuse, Bostrom, Schmidhuber, Musk)

Languages and Systems



[fig. from Frank Wood]

George E.P. Box (1919 - 2013)

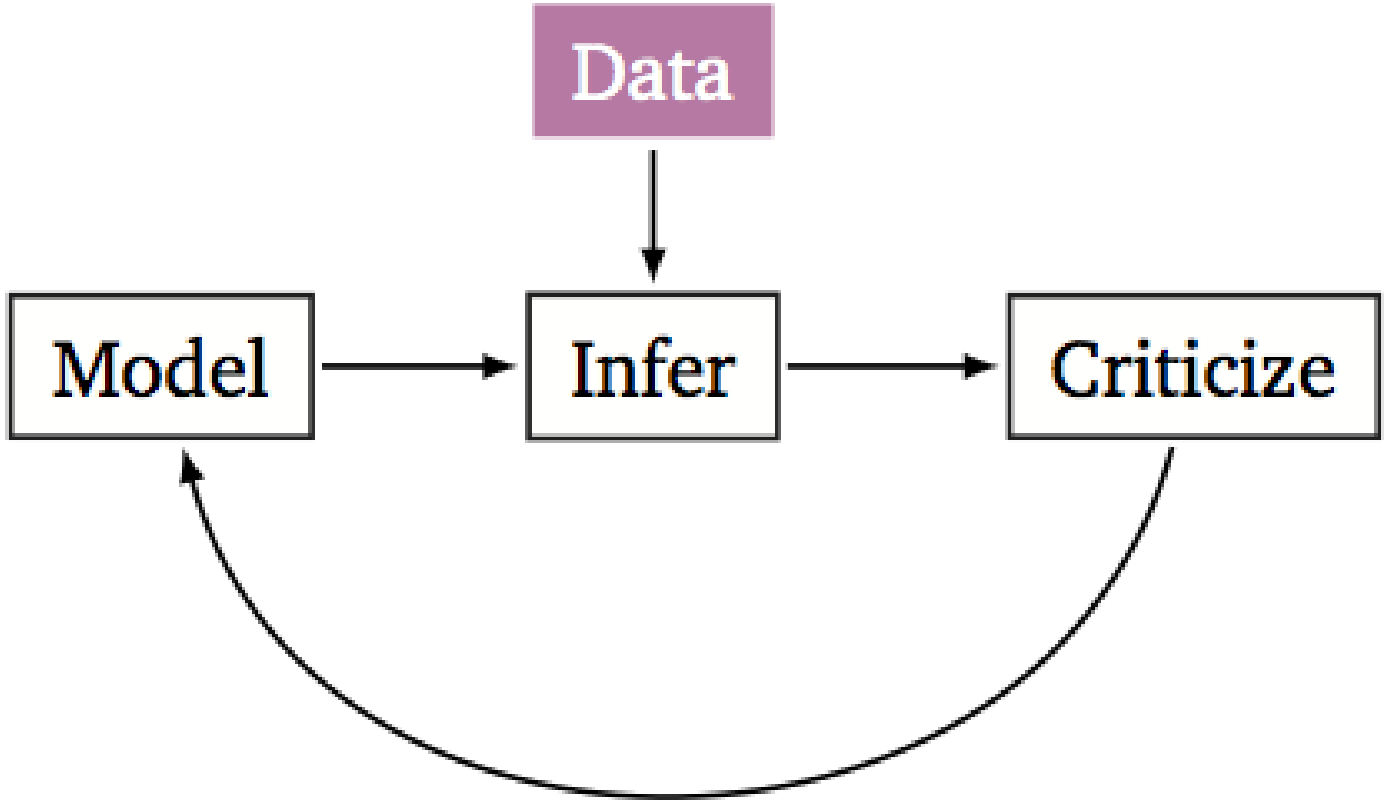


An iterative process for science:

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

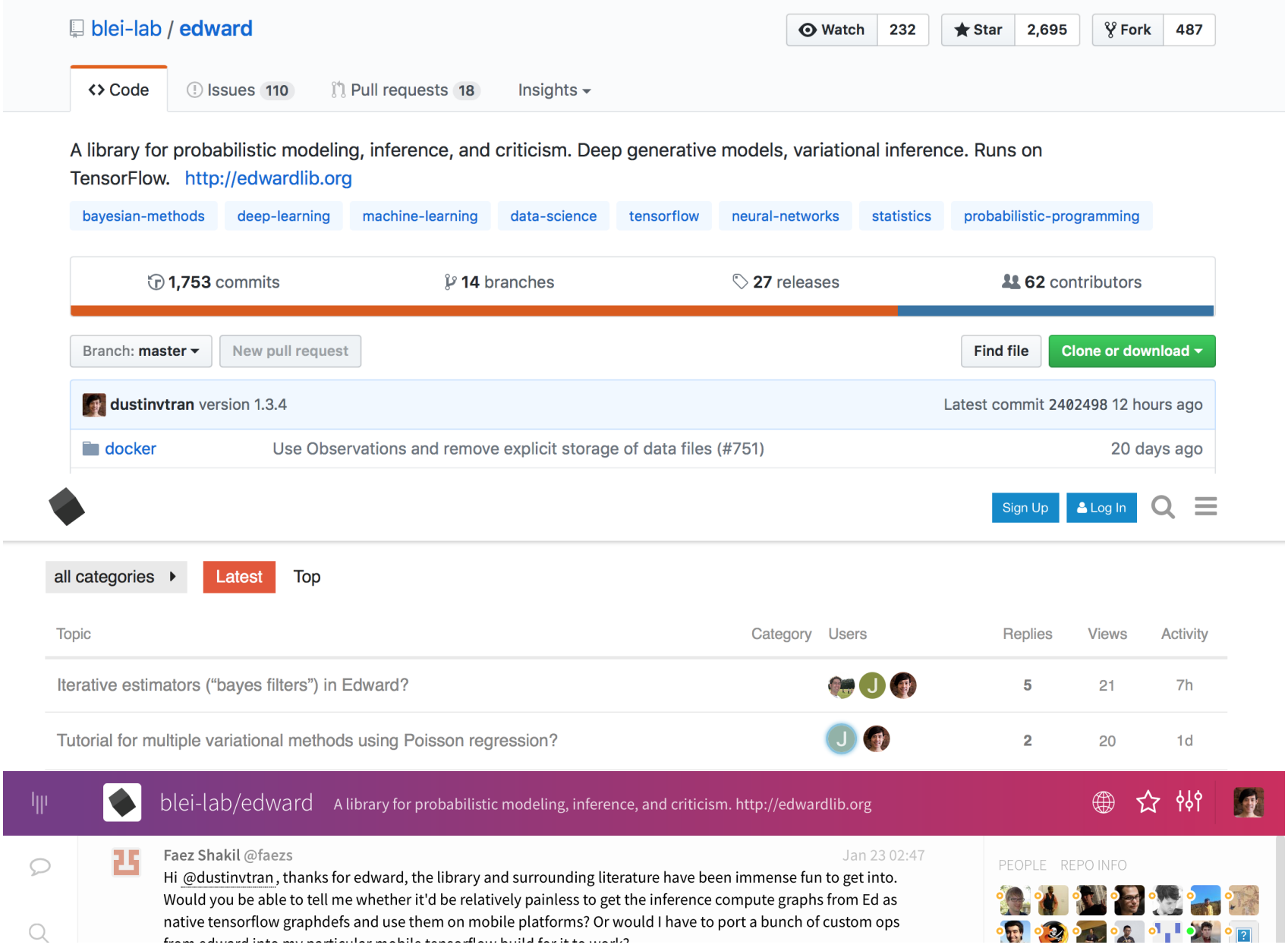
[Box & Hunter 1962, 1965; Box & Hill 1967; Box 1976, 1980]

Box's Loop



Edward is a library designed around this loop.

[Box 1976, 1980; Blei 2014]



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

How do we use Edward?

Model

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

```
# univariate normal
Normal(loc=0.0, scale=1.0)
# vector of 5 univariate normals
Normal(loc=tf.zeros(5), scale=tf.ones(5))
# 2 x 3 matrix of Exponentials
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 $\mathbf{x}^*$, $\mathbf{x}^* \sim p(\mathbf{x} \mid \theta^*)$.

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

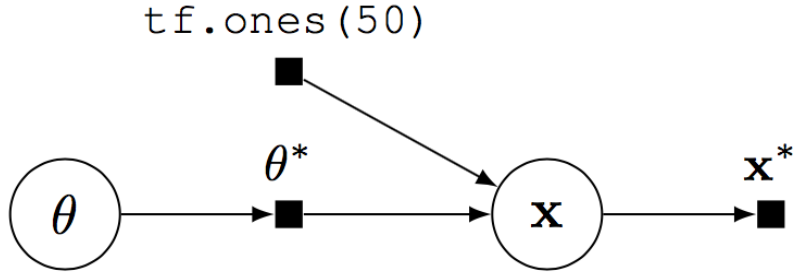
Example: Beta-Bernoulli

Consider a Beta-Bernoulli model,

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

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

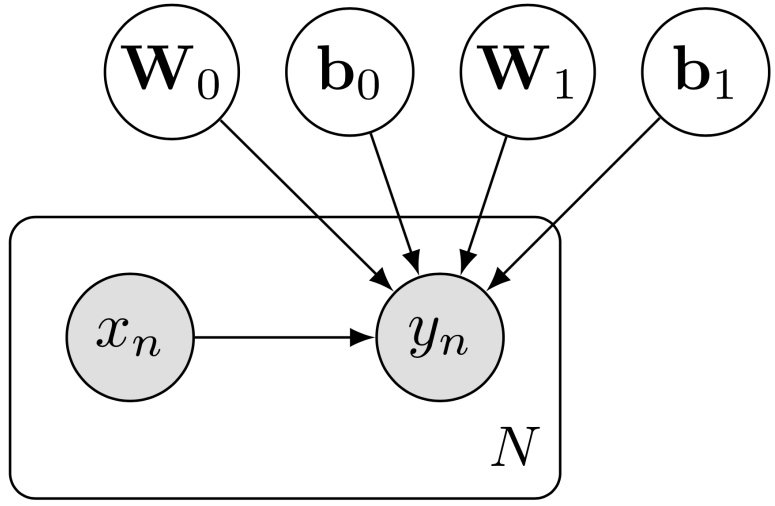
```
1  theta = Beta(1.0, 1.0)
2  x = Bernoulli(probs=tf.ones(50) * theta)
```



Fetching **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.

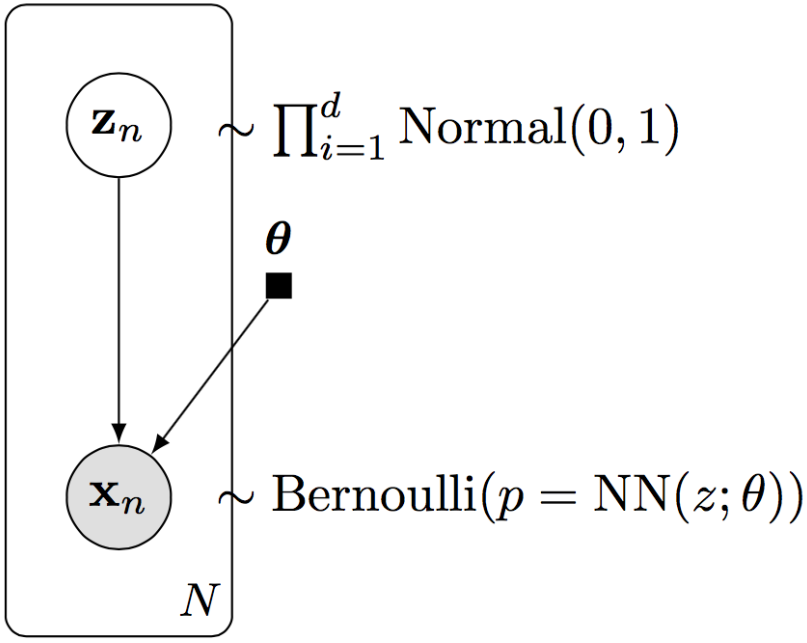
Example: Bayesian neural network for classification

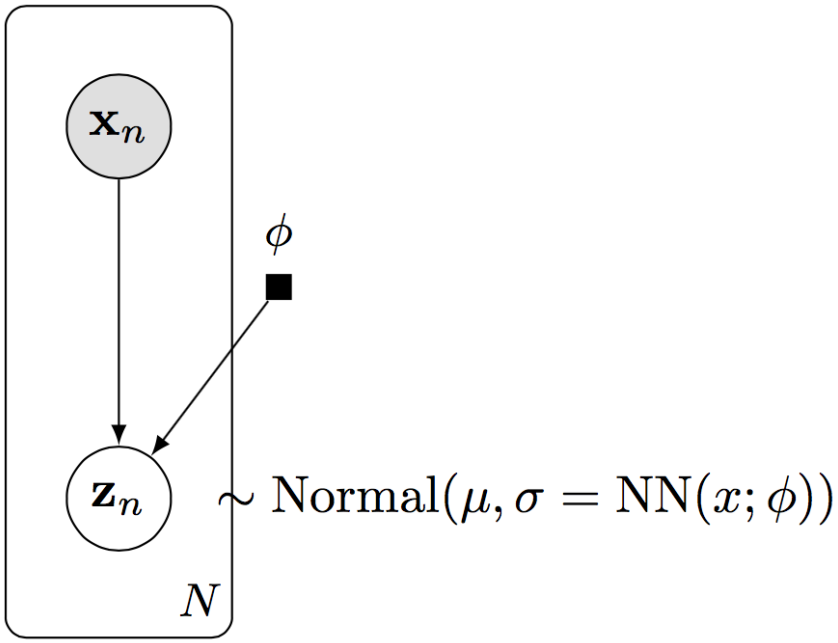



```
W_0 = Normal(mu=tf.zeros([D, H]), sigma=tf.ones([D, H]))
W_1 = Normal(mu=tf.zeros([H, 1]), sigma=tf.ones([H, 1]))
b_0 = Normal(mu=tf.zeros(H), sigma=tf.ones(L))
b_1 = Normal(mu=tf.zeros(1), sigma=tf.ones(1))

x = tf.placeholder(tf.float32, [N, D])
y = Bernoulli(logits=tf.matmul(tf.nn.tanh(tf.matmul(x, W_0) + b_0), W_1) + b_1)
```

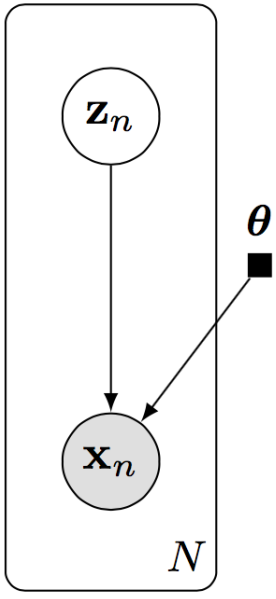
Example: Variational Auto-Encoder for Binarized MNIST



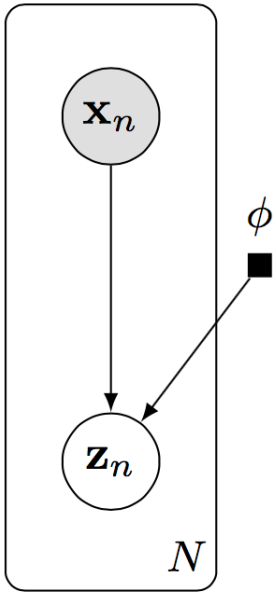


[Kingma & Welling 2014; Rezende+ 2014]

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))
```

[Kingma & Welling 2014; Rezende+ 2014]

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 **z**, β, binding model variables to approximate factors;
2. observed variables **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

```
N = 1000  # number of data points
d = 10  # latent dimensionality

# DATA
x_data = np.loadtxt('mnist.txt', np.float32)

# 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))

# INFERENCE
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 = ed.KLqp({z: qz}, data={x: x_data, qx: x_data})
inference.run()
```

Example: Variational Auto-Encoder for Binarized MNIST

```
N = 1000  # number of data points
d = 10  # latent dimensionality

# DATA
x_data = np.loadtxt('mnist.txt', np.float32)

# 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))

# INFERENCE
T = 10000  # number of samples
qz = Empirical(params=tf.Variable(tf.zeros([T, N, d])))

inference = ed.HMC({z: qz}, data={x: x_data})
inference.run()
```

Non-Bayesian Methods: GANs

GANs posit a generative process,

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

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.

[Goodfellow+ 2014]

Non-Bayesian Methods: GANs

```
def generative_network(eps):
  h = Dense(256, activation='relu')(eps)
  return Dense(28 * 28, activation=None)(h)

def discriminative_network(x):
  h = Dense(28 * 28, activation='relu')(x)
  return Dense(h, activation=None)(1)

# Probabilistic model
eps = Normal(loc=tf.zeros([N, d]), scale=tf.ones([N, d]))
x = generative_network(eps)

inference = ed.GANInference(data={x: x_train},
    discriminator=discriminative_network)
inference.run()
```

[Goodfellow+ 2014]

Non-Bayesian Methods: GANs

```
def generative_network(eps):
  h = Dense(256, activation='relu')(eps)
  return Dense(28 * 28, activation=None)(h)

def discriminative_network(x):
  h = Dense(28 * 28, activation='relu')(x)
  return Dense(h, activation=None)(1)

# Probabilistic model
eps = Normal(loc=tf.zeros([N, d]), scale=tf.ones([N, d]))
x = generative_network(eps)

inference = ed.WGANInference(data={x: x_train},
    discriminator=discriminative_network)
inference.run()
```

[Arjovsky+ 2017; Gulrajani+ 2017]

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 (≈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).

[in preparation]

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. Distributed probabilistic programming. (Google)
2. Idiomatic probabilistic programming with TensorFlow. (Google)
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.