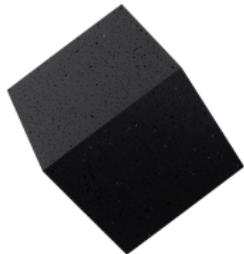


# **Edward: A library for probabilistic modeling, inference, and criticism**

Dustin Tran  
Columbia University





Alp Kucukelbir



Adji Dieng



Maja Rudolph



Dawen Liang



David Blei



Matt Hoffman



Kevin Murphy



Eugene Brevdo

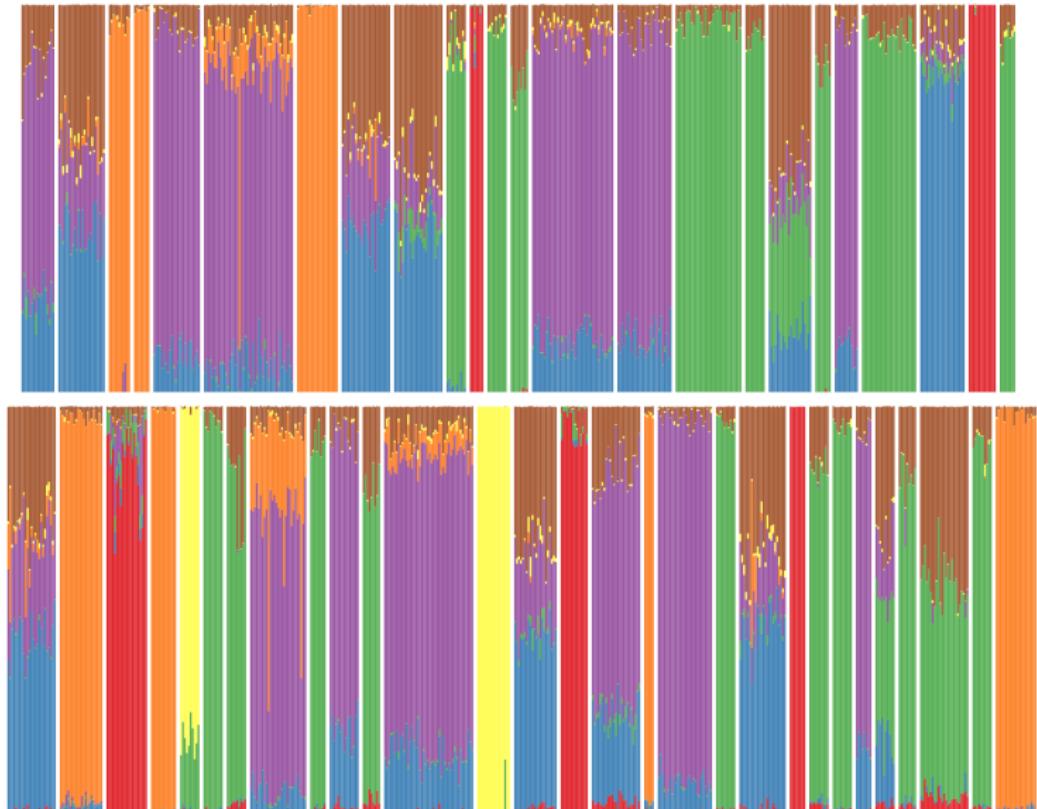


Rif Saurous

<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
Game Season Team Coach Play Points Games Giants Second Players	Life Know School Street Man Family Says House Children Night	Film Movie Show Life Television Films Director Man Story Says	Book Life Books Novel Story Man Author House War Children	Wine Street Hotel House Room Night Place Restaurant Park Garden
<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
Bush Campaign Clinton Republican House Party Democratic Political Democrats Senator	Building Street Square Housing House Buildings Development Space Percent Real	Won Team Second Race Round Cup Open Game Play Win	Yankees Game Mets Season Run League Baseball Team Games Hit	Government War Military Officials Iraq Forces Iraqi Army Troops Soldiers
<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>
Children School Women Family Parents Child Life Says Help Mother	Stock Percent Companies Fund Market Bank Investors Funds Financial Business	Church War Women Life Black Political Catholic Government Jewish Pope	Art Museum Show Gallery Works Artists Street Artist Paintings Exhibition	Police Yesterday Man Officer Officers Case Found Charged Street Shot

Topics found in 1.8M articles from the New York Times

(Hoffman et al. 2013)



Population analysis of 2 billion genetic measurements

(Gopalan et al. 2014)



Analysis of 1.7M taxi trajectories, in Stan

(Kucukelbir et al. 2016)

# Challenges in Probabilistic Programming

1. **Frameworks.** What principles should guide the design of a probabilistic programming language?
2. **Modeling languages.** How do we expose structure in probabilistic programs?
3. **Inference languages.** How do we build infrastructure to support a variety of algorithms?

What principles should guide the design of a probabilistic  
programming language?

# Why are guiding principles important?

## Classification

Identifying to which category an object belongs to.

**Applications:** Spam detection, Image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, ...

[— Examples](#)

## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, ridge regression, Lasso, ...

[— Examples](#)

## Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, ...

[— Examples](#)

## Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency

**Algorithms:** PCA, feature selection, non-negative matrix factorization.

[— Examples](#)

## Model selection

Comparing, validating and choosing parameters and models.

**Goal:** Improved accuracy via parameter tuning

**Modules:** grid search, cross validation, metrics.

[— Examples](#)

## Preprocessing

Feature extraction and normalization.

**Application:** Transforming input data such as text for use with machine learning algorithms.

**Modules:** preprocessing, feature extraction.

[— Examples](#)

Is this the right organization and set of abstractions?

(Source: scikit-learn)

# George E.P. Box (1919 - 2013)

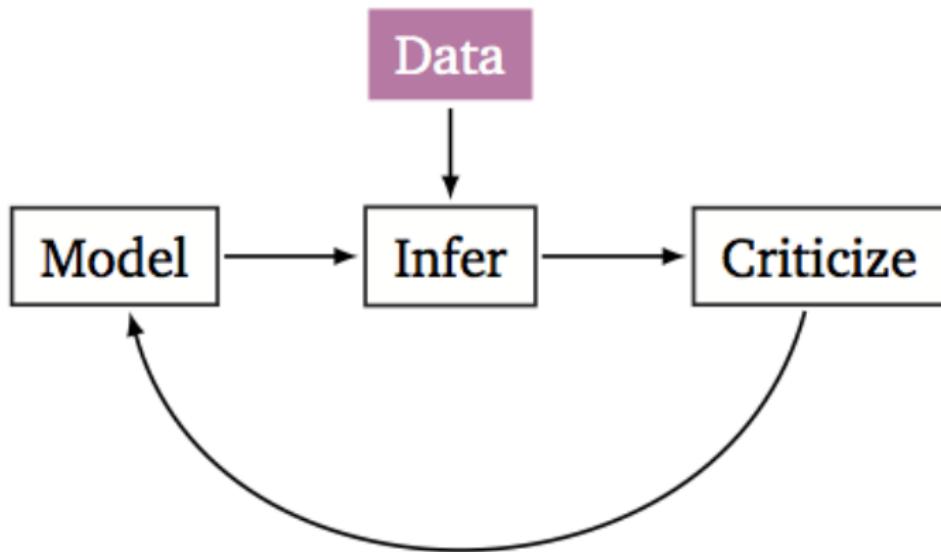


An iterative process for science:

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; Box, 1980; Blei, 2014)

# Computational Graphs

Computational graphs are graphs where **nodes** represent operations and **edges** represent tensors communicated between operations.

Computational graphs enable useful numerical tools.

Such tools include partial execution, distributed execution, automatic differentiation, and many more.

## Example: Modeling Coin Flips

```
# DATA
x_data = np.array([0, 1, 0, 0, 0, 0, 0, 0, 0, 1])

# MODEL
p = Beta(a=1.0, b=1.0)
x = Bernoulli(p=tf.ones(10) * p)

# INFERENCE
qp_a = tf.nn.softplus(tf.Variable(0.0))
qp_b = tf.nn.softplus(tf.Variable(0.0))
qp = Beta(a=qp_a, b=qp_b)

inference = ed.KLqp({p: qp}, data={x: x_data})
inference.run(n_iter=500)

# CRITICISM
x_post = ed.copy(x, {p: qp})
T = lambda xs, zs: tf.reduce_mean(xs[x_post])
ed.ppc(T, data={x_post: x_data})
```

How do we expose structure in probabilistic programs?

# Why is structure important?



Topics found in 1.8M articles from the New York Times

What existing probabilistic programming languages enable this analysis?

# Why is structure important?



Topics found in 1.8M articles from the New York Times

What existing probabilistic programming languages enable this analysis?

Language	Inference
Church, Venture, Anglican	SMC, MH
Stan	ADVI (w/ mini-batches)
WebPPL, PyMC3	ADVI (w/ mini-batches and inference networks)
Infer.NET	VMP

**Punchline:** We need the graph structure (and flexible approximations).

# Model: Random Variables

A random variable  $\mathbf{x}$  is an *object* parameterized by tensors  $\theta^*$ .

```
# 1 univariate Gaussian
Normal(mu=tf.constant(0.0), sigma=tf.constant(1.0))
# 2 x 3 matrix of Exponentials
Exponential(lam=tf.ones([2, 3]))
# 1 K-dimensional Dirichlet
Dirichlet(alpha=np.array([0.1]*K)
```

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

A random variable wraps a tensor  $\mathbf{x}^*$ , where  $\mathbf{x}^* \sim p(\mathbf{x} \mid \theta^*)$  is a sample.

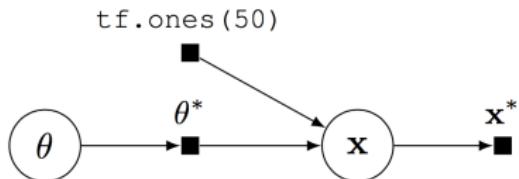
## Example: Beta-Bernoulli

Consider a Beta-Bernoulli model,

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

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

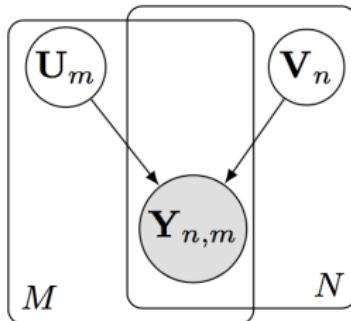
```
1 theta = Beta(a=1.0, b=1.0)
2 x = Bernoulli(p=tf.ones(50) * theta)
```



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

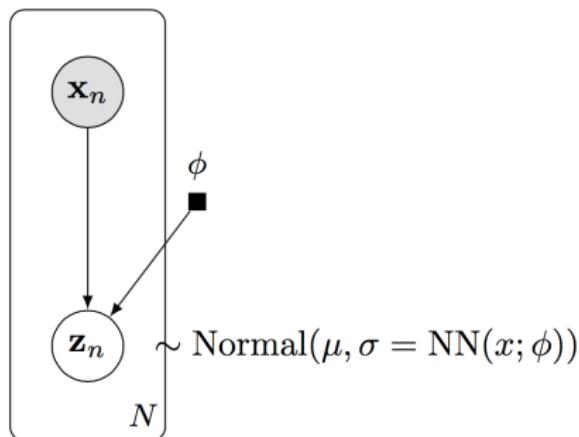
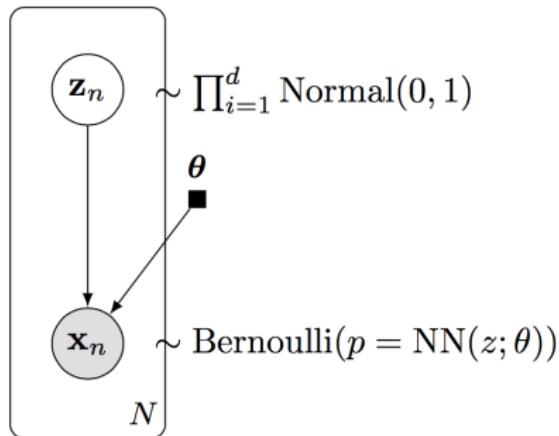
All computation is represented on the graph, enabling us to leverage model structure during inference.

# Example: Gaussian Matrix Factorization



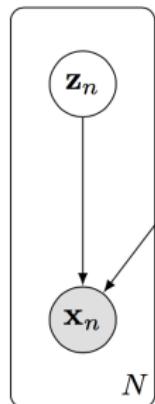
```
1 N = 10
2 M = 10
3 K = 5 # latent dimension
4
5 U = Normal(mu=tf.zeros([M, K]), sigma=tf.ones([M, K]))
6 V = Normal(mu=tf.zeros([N, K]), sigma=tf.ones([N, K]))
7 Y = Normal(mu=tf.matmul(U, V, transpose_b=True), sigma=tf.ones([N, M]))
```

## Example: Variational Auto-Encoder for Binarized MNIST

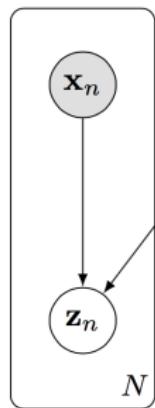


(Kingma & Welling, 2014; Rezende et al., 2014)

# Example: Variational Auto-Encoder for Binarized MNIST



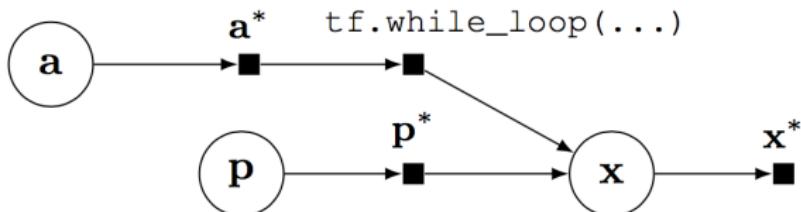
```
# Generative model  
z = Normal(mu=tf.zeros([N, d]), sigma=tf.ones([N, d]))  
h = slim.fully_connected(z, 256, activation_fn=tf.nn.relu)  
x = Bernoulli(logits=slim.fully_connected(h, 28 * 28))
```



```
# Variational model  
qx = tf.placeholder(tf.float32, [N, 28 * 28])  
qh = slim.fully_connected(qx, 256, activation_fn=tf.nn.relu)  
qz = Normal(mu=slim.fully_connected(qh, d),  
            sigma=slim.fully_connected(qh, d, activation_fn=tf.nn.softplus))
```

(Kingma & Welling, 2014; Rezende et al., 2014)

## Model: BNP & Probabilistic Programs



Random variables can also be placed in control flow operations, enabling models with a Dirichlet process.

Importantly, the computational graph provides an elegant way of teasing out static conditional dependence structure (**p**) from dynamic dependence structure (**a**).

How do we build infrastructure to support a variety of algorithms?

# Inference: Representing Classes

Consider a model  $p(\mathbf{x}, \mathbf{z}, \beta)$  of data  $\mathbf{x}_{\text{train}}$  with latent variables  $(\mathbf{z}, \beta)$ . The goal is to calculate the posterior,

$$q(\mathbf{z}, \beta) \approx p(\mathbf{z}, \beta \mid \mathbf{x}_{\text{train}}),$$

using an approximating family,  $q(\mathbf{z}, \beta; \lambda)$ .

In Edward, this is

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

All inference has two **inputs**:

- (1) latent variables  $\mathbf{z}, \beta$ , binding model variables to approximate factors;
- (2) observed variables  $\mathbf{x}$ , binding model variables to data.

Inference has methods, e.g., `run()` runs the algorithm.

# Inference: Representing Classes

Variational inference. We build the variational family in the graph.

```
1 qbeta = Normal(mu=tf.Variable(tf.zeros([K, D])),  
2                 sigma=tf.exp(tf.Variable(tf.zeros([K, D])))  
3 qz = Categorical(logits=tf.Variable(tf.zeros([N, K])))  
4  
5 inference = ed.VariationalInference({beta: qbeta, z: qz}, data={x: x_train})
```

Monte Carlo. The approximating family uses Empirical random variables.

```
1 T = 10000 # number of samples  
2 qbeta = Empirical(params=tf.Variable(tf.zeros([T, K, D])))  
3 qz = Empirical(params=tf.Variable(tf.zeros([T, N])))  
4  
5 inference = ed.MonteCarlo({beta: qbeta, z: qz}, data={x: x_train})
```

We are also working on exact inference.

# Inference: Composing Inference

Core to Edward's design is that inference can be written as a collection of separate inference programs.

For example, here is variational EM.

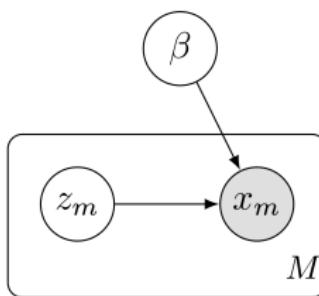
```
1 qbeta = PointMass(params=tf.Variable(tf.zeros([K, D])))
2 qz = Categorical(logits=tf.Variable(tf.zeros[N, K]))
3
4 inference_e = ed.VariationalInference({z: qz}, data={x: x_data, beta: qbeta})
5 inference_m = ed.MAP({beta: qbeta}, data={x: x_data, z: qz})
6
7 for _ in range(10000):
8     inference_e.update()
9     inference_m.update()
```

We can also write message passing algorithms, which work over a collection of local inference problems. This includes for example expectation propagation.

# Inference: Data Subsampling

Stochastic optimization scales inference to massive data (Welling and Teh, 2011; Hoffman et al., 2013).

To support such algorithms, we represent only a subgraph of the full model.



We then pass in a scale argument during initialization,

```
inference = ed.VariationalInference({beta: qbeta, z: qz},  
                                     data={x: x_batch})  
inference.initialize(scale={x: float(N) / M, z: float(N) / M})
```

# Summary

1. Leveraged Box's loop as the design principle of Edward.
2. Developed a probabilistic programming language on computational graphs, with model structure exposed to the user.
3. Developed a language around inference, including both model-specific and generic algorithms.

# Key References

- **D. Tran**, A. Kucukelbir, A. Dieng, M. Rudolph, D. Liang, and D.M. Blei.  
Edward: A library for probabilistic modeling, inference, and criticism.  
[edwardlib.org](http://edwardlib.org)  
(arXiv preprint arXiv:1610.09787)
- **D. Tran**, M. Hoffman, K. Murphy, E. Brevdo, R. Saurous, and D.M. Blei.  
Deep probabilistic programming.  
(ICLR submission)