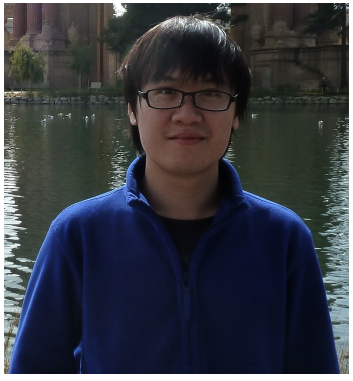
Edward: A library for probabilistic modeling, inference, and criticism

Dustin Tran
Columbia University

Alp Kucukelbir

Adji Dieng

Maja Rudolph

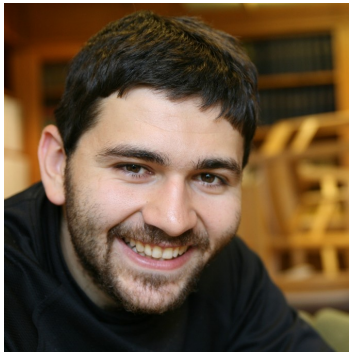


Dawen Liang

David Blei

Matt Hoffman

Kevin Murphy

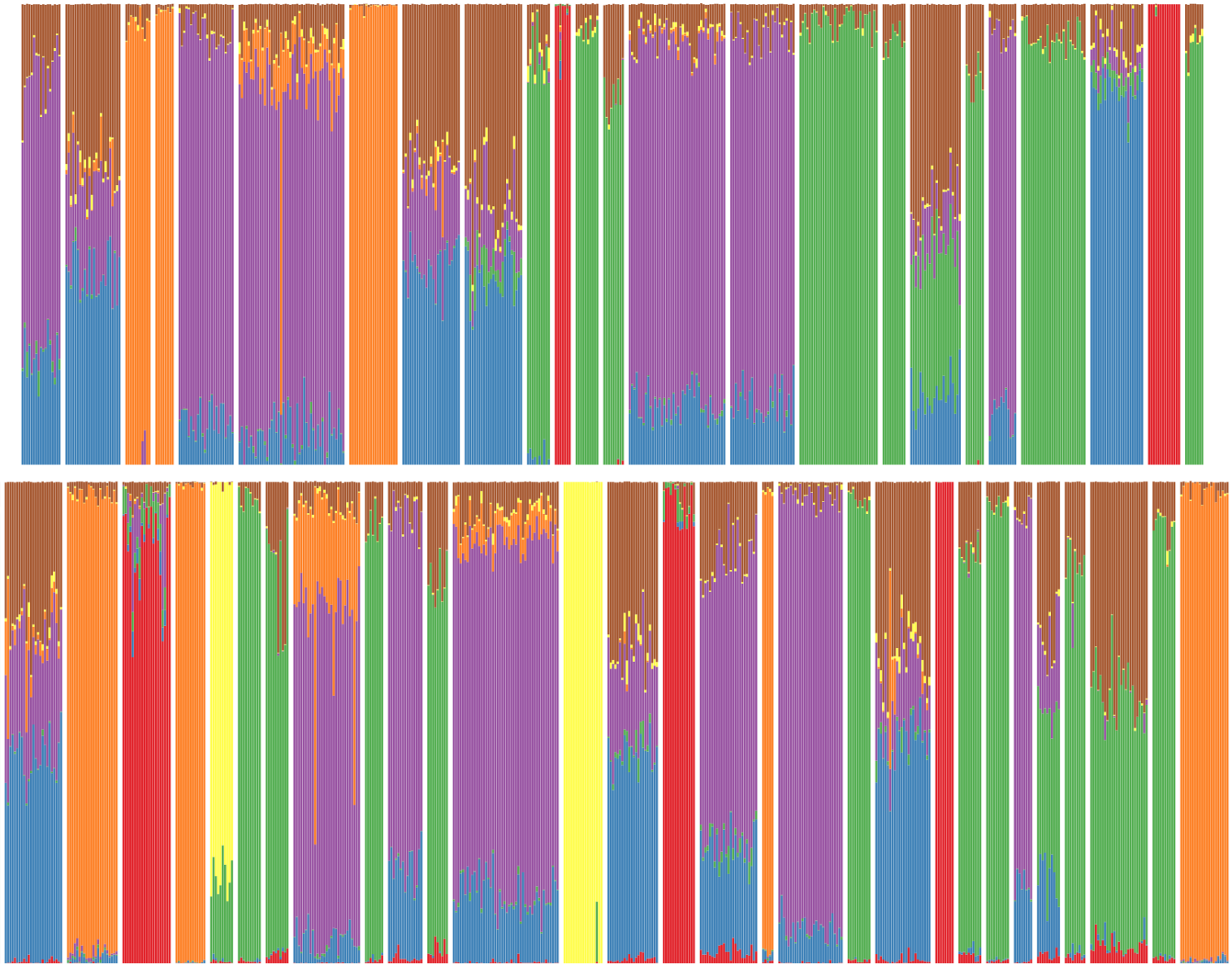


Eugene Brevdo

Rif Saurous

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 easily develop algorithms and support a large class of them?

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

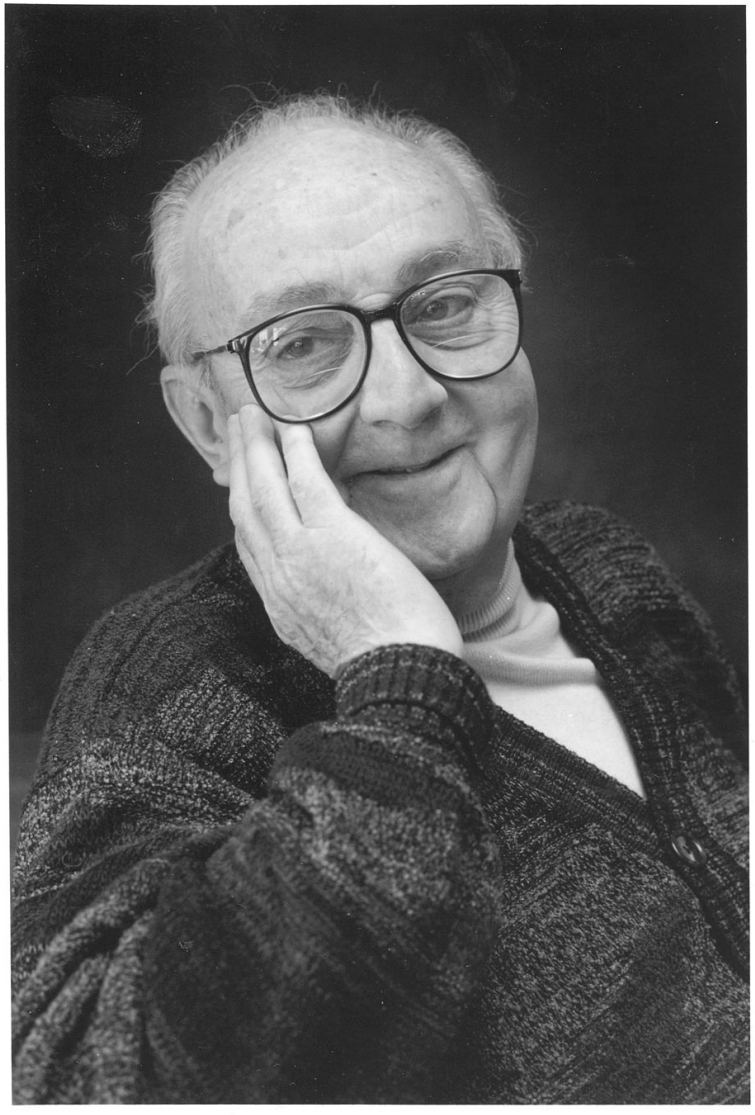
Feature extraction and normalization.

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)

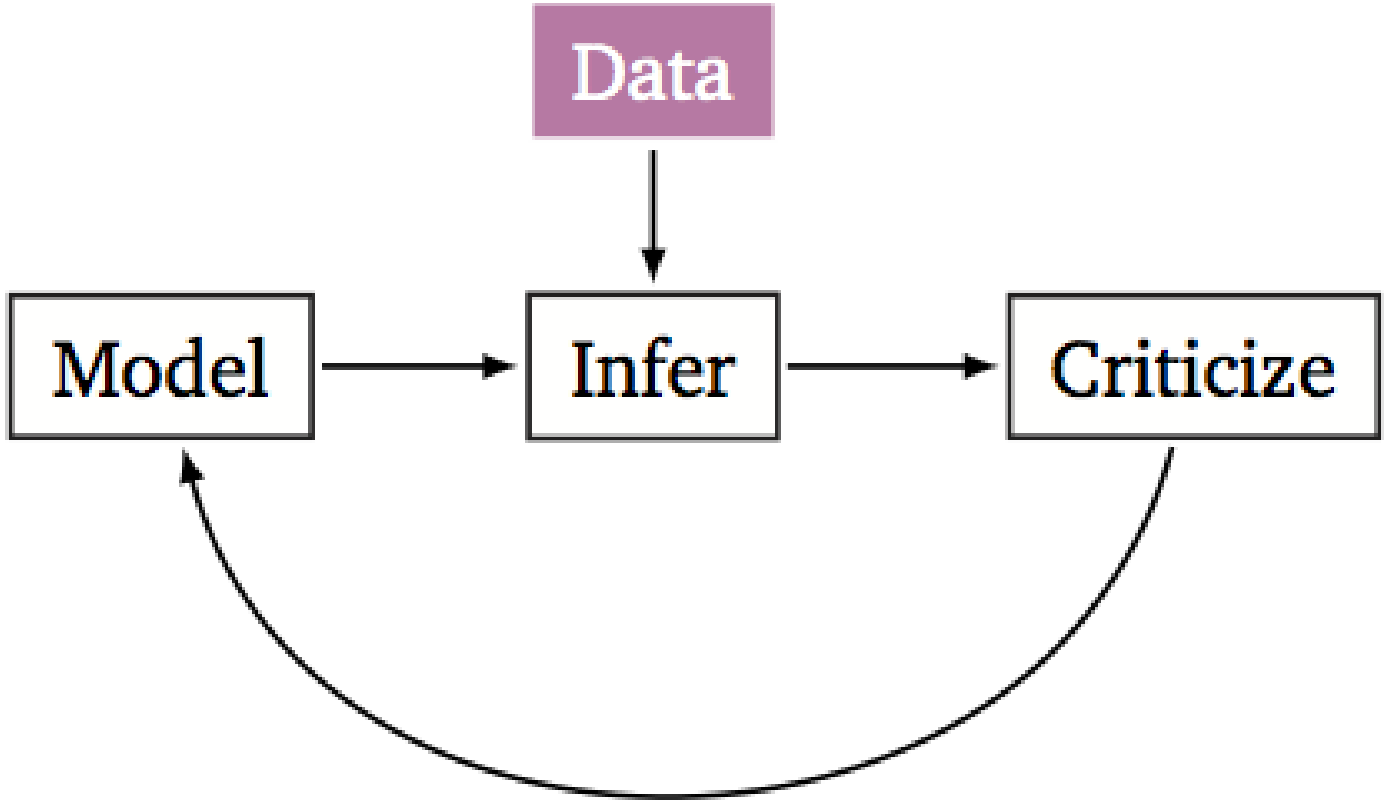


An iterative process for science:

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)

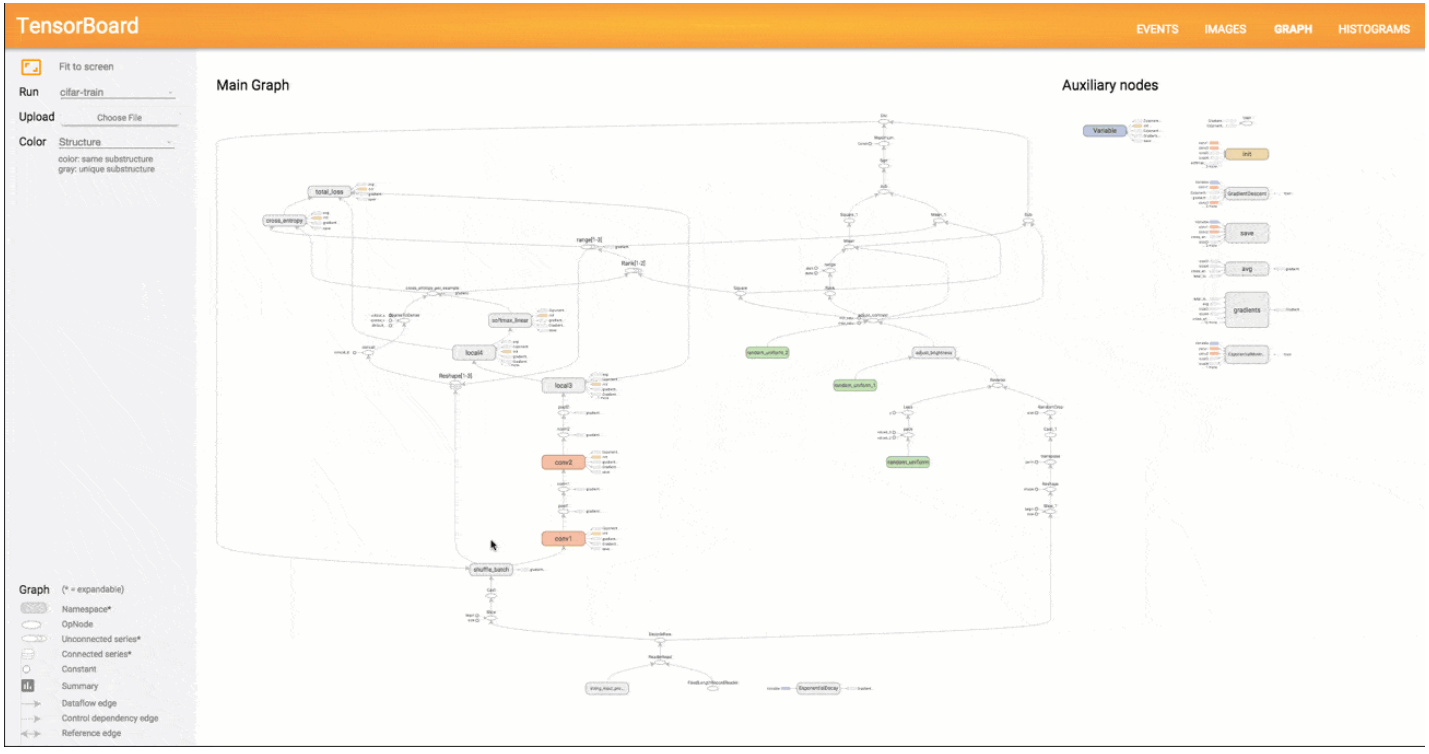
Box's Loop



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.

They speed up computation with hardware such as GPUs, scale up computation with distributed training, and simplify engineering effort with automatic differentiation.

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?

1	2	3	4	5
Game	Life	Film	Book	Wine
Season	Know	Movie	Life	Street
Team	School	Show	Books	Hotel
Coach	Street	Life	Novel	House
Play	Man	Television	Story	Room
Points	Family	Films	Man	Night
Games	Says	Director	Author	Place
Giants	House	Man	House	Restaurant
Second	Children	Story	War	Park
Players	Night	Says	Children	Garden

6	7	8	9	10
Bush	Building	Won	Yankees	Government
Campaign	Street	Team	Game	War
Clinton	Square	Second	Mets	Military
Republican	Housing	Race	Season	Officials
House	House	Round	Run	Iraq
Party	Buildings	Cup	League	Forces
Democratic	Development	Open	Baseball	Iraqi
Political	Space	Game	Team	Army
Democrats	Percent	Play	Games	Troops
Senator	Real	Win	Hit	Soldiers

11	12	13	14	15
Children	Stock	Church	Art	Police
School	Percent	War	Museum	Yesterday
Women	Companies	Women	Show	Man
Family	Fund	Life	Gallery	Officer
Parents	Market	Black	Works	Officers
Child	Bank	Political	Artists	Case
Life	Investors	Catholic	Street	Found
Says	Funds	Government	Artist	Charged
Help	Financial	Jewish	Paintings	Street
Mother	Business	Pope	Exhibition	Shot

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

```
# 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()`.

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 $p(\mathbf{x}; \theta)$.

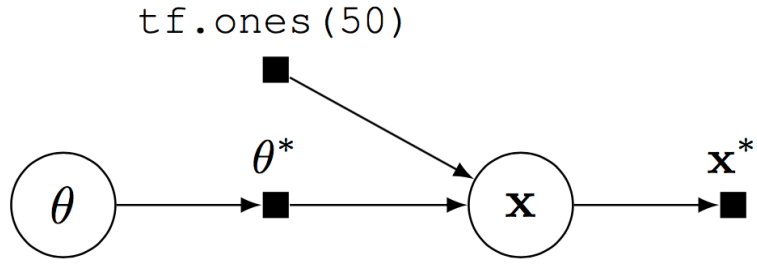
Example: Beta-Bernoulli

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 $\mathbf{x} \in \{0, 1\}^{50}$.

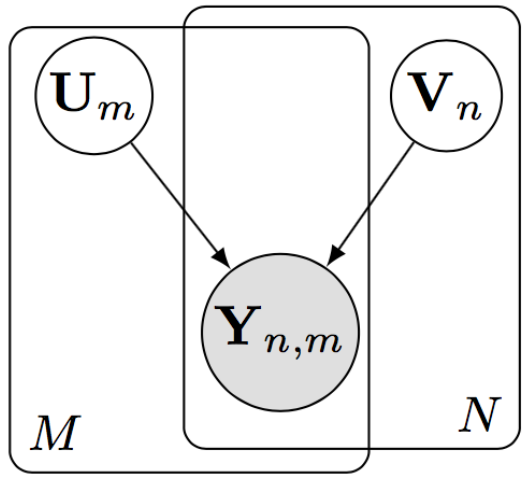
```
1  theta = Beta(a=1.0, b=1.0)
2  x = Bernoulli(p=tf.ones(50) * theta)
```



Fetching **x** from the graph generates a binary vector of 50 elements.

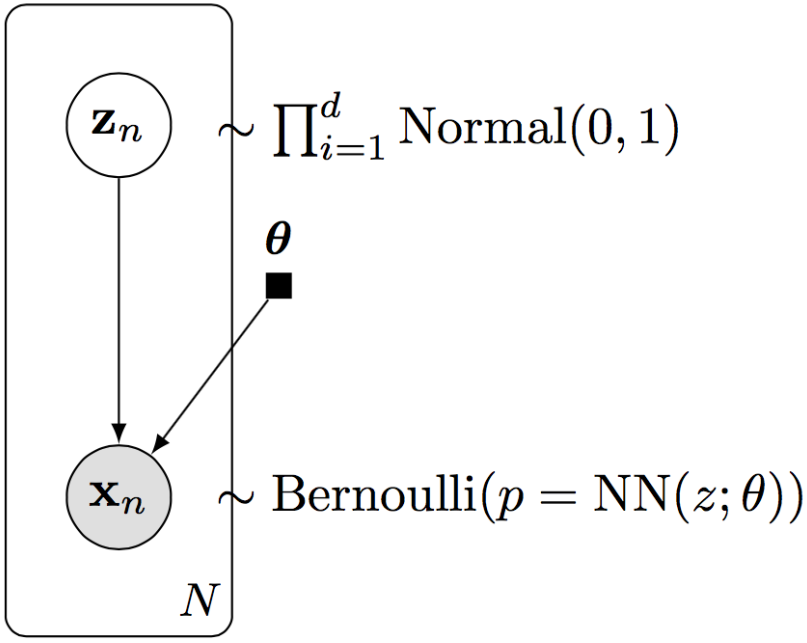
All computation is represented on the graph, enabling us to leverage model structure during inference.

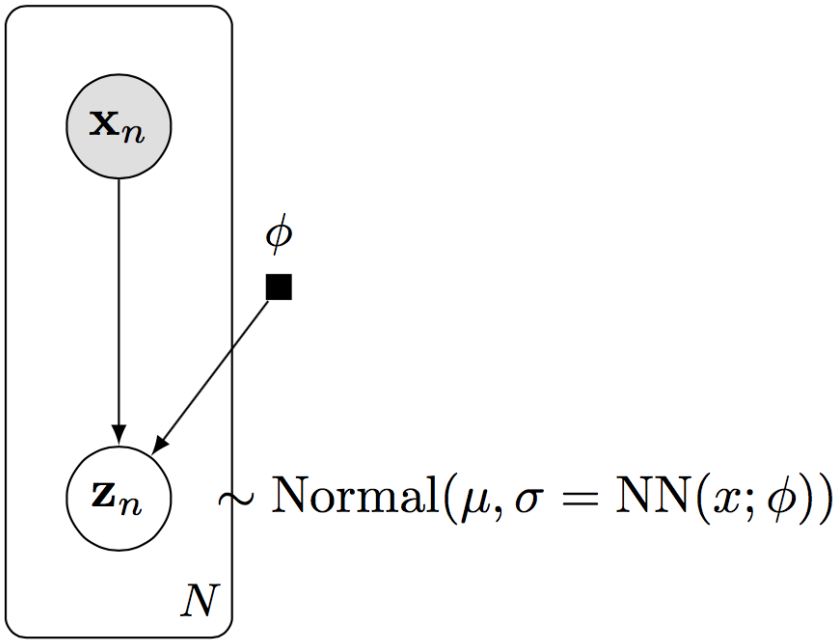
Example: Gaussian Matrix Factorization




```
N = 10
M = 10
K = 5  # latent dimension

U = Normal(mu=tf.zeros([M, K]), sigma=tf.ones([M, K]))
V = Normal(mu=tf.zeros([N, K]), sigma=tf.ones([N, K]))
Y = Normal(mu=tf.matmul(U, V, transpose_b=True), sigma=tf.ones([N, M]))
```

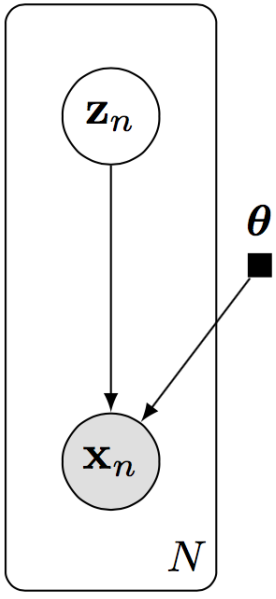
Example: Variational Auto-Encoder for Binarized MNIST





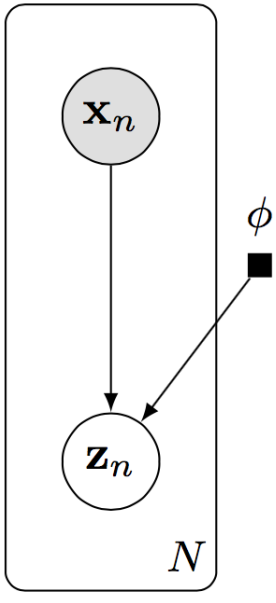
(Kingma & Welling, 2014; Rezende et al., 2014)

Example: Variational Auto-Encoder for Binarized MNIST



$\boldsymbol{\theta}$

```
# 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))
```

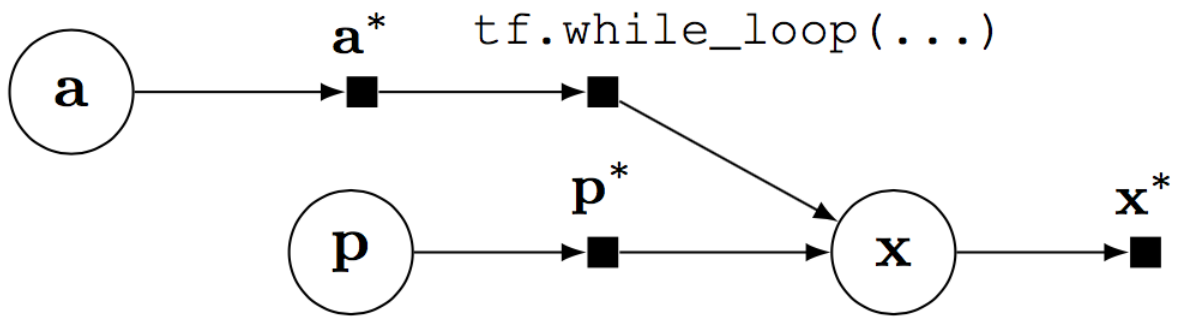


ϕ

```
# 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 easily develop algorithms and support a large class of them?

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.

```
qbeta = Normal(mu=tf.Variable(tf.zeros([K, D])),
               sigma=tf.exp(tf.Variable(tf.zeros[K, D])))
qz = Categorical(logits=tf.Variable(tf.zeros[N, K]))

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

Monte Carlo. The approximating family uses Empirical random variables.

```
T = 10000  # number of samples
qbeta = Empirical(params=tf.Variable(tf.zeros([T, K, D])))
qz = Empirical(params=tf.Variable(tf.zeros([T, N])))

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.

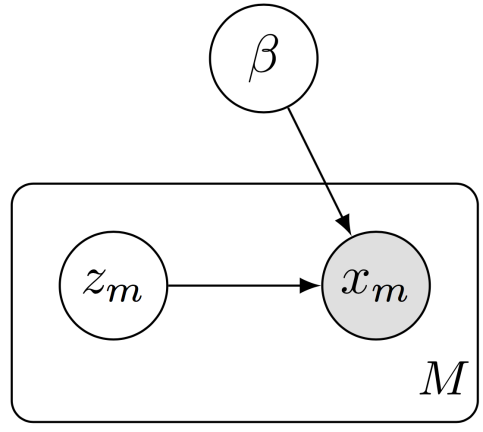
```
qbeta = PointMass(params=tf.Variable(tf.zeros([K, D])))
qz = Categorical(logits=tf.Variable(tf.zeros[N, K]))

inference_e = ed.VariationalInference({z: qz}, data={x: x_data, beta: qbeta})
inference_m = ed.MAP({beta: qbeta}, data={x: x_data, z: qz})

for _ in range(10000):
  inference_e.update()
  inference_m.update()
```

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

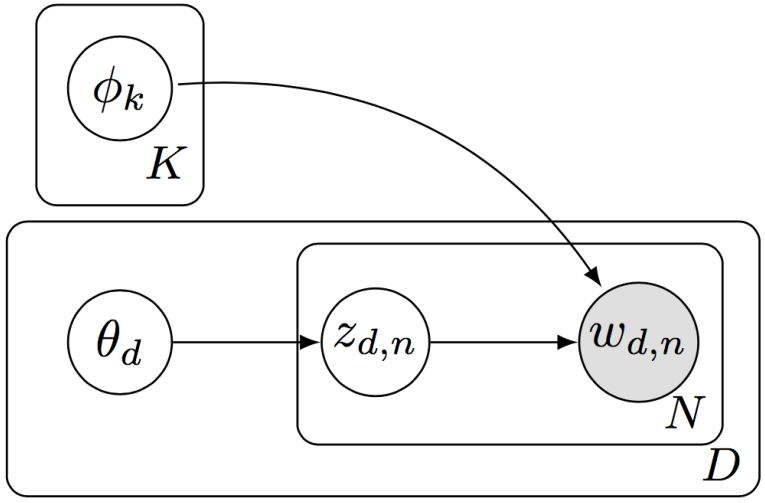
1. Leveraged Box's loop as the design principle of Edward.
2. Developed a probabilistic programming language on computational graphs, with model structure exposed to the user.
3. Developed a language around inference, including both model-specific and generic algorithms.

References

edwardlib.org

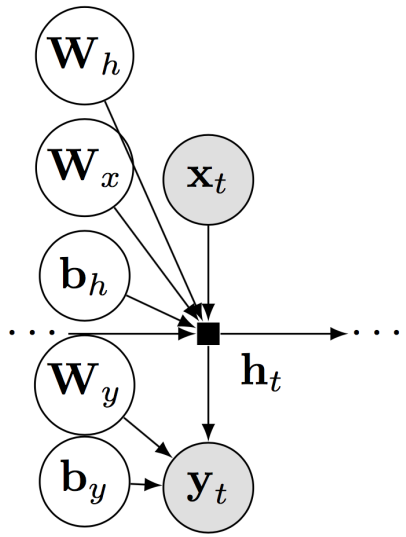
- **D. Tran**, A. Kucukelbir, A. Dieng, M. Rudolph, D. Liang, and D.M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. (arXiv preprint arXiv:1610.09787)
- **D. Tran**, M.D. Hoffman, R.A. Saurous, E. Brevdo, K. Murphy, and D.M. Blei. Deep probabilistic programming. (arXiv preprint arXiv:1701.03757)

Example: Latent Dirichlet allocation



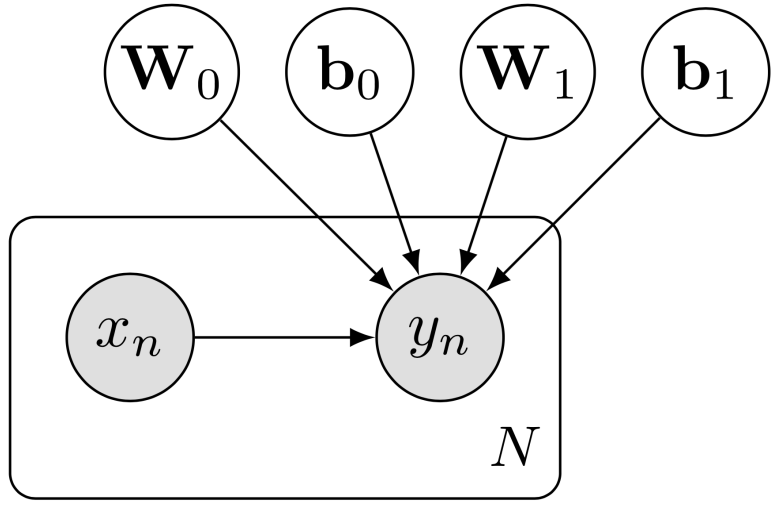

```
D = 50  # number of documents
N = [11502, 213, 1523, 1351, ...]  # words per doc
K = 10  # number of topics
V = 100000  # vocabulary size

theta = Dirichlet(alpha=tf.zeros([D, K]) + 0.1)
phi = Dirichlet(alpha=tf.zeros([K, V]) + 0.05)
z = [[0] * N] * D
w = [[0] * N] * D
for d in range(D):
  for n in range(N[d]):
    z[d][n] = Categorical(pi=theta[d, :])
    w[d][n] = Categorical(pi=phi[z[d][n], :])
```

Example: Bayesian recurrent neural network




```
1   def rnn_cell(hprev, xt):
2     return tf.tanh(tf.dot(hprev, Wh) + tf.dot(xt, Wx) + bh)
3
4   Wh = Normal(mu=tf.zeros([H, H]), sigma=tf.ones([H, H]))
5   Wx = Normal(mu=tf.zeros([D, H]), sigma=tf.ones([D, H]))
6   Wy = Normal(mu=tf.zeros([H, 1]), sigma=tf.ones([H, 1]))
7   bh = Normal(mu=tf.zeros(H), sigma=tf.ones(H))
8   by = Normal(mu=tf.zeros(1), sigma=tf.ones(1))
9
10  x = tf.placeholder(tf.float32, [None, D])
11  h = tf.scan(rnn_cell, x, initializer=tf.zeros(H))
12  y = Normal(mu=tf.matmul(h, Wy) + by, sigma=1.0)
```

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)
```

Experiment: GPU-accelerated Hamiltonian Monte Carlo

```
# Model
x = tf.Variable(x_data, trainable=False)
beta = Normal(mu=tf.zeros(D), sigma=tf.ones(D))
y = Bernoulli(logits=tf.dot(x, beta))

# Inference
qbeta = Empirical(params=tf.Variable(tf.zeros([T, D])))
inference = ed.HMC({beta: qbeta}, data={y: y_data})
inference.run(step_size=0.5 / N, n_steps=100)
```

We perform inference on Bayesian logistic regression for the Covertype dataset ($N = 581012$, $D = 54$). We use a 12-core Intel i7-5930K CPU at 3.50GHz and a NVIDIA Titan X (Maxwell) GPU.

We compare the runtime of HMC for $T = 100$ iterations (and same settings).

Probabilistic programming language	Runtime
Stan (1 CPU) [2]	171 sec
PyMC3 (12 CPU) [6]	361 sec
Edward (12 CPU)	**8.2 sec**
Edward (GPU)	**4.9 sec** (35x faster than Stan)

Experiment: Recent Methods in Variational Inference

Inference method	Negative log-likelihood
Variational auto-encoder (VAE) [3]	≤ 88.2
VAE without analytic KL	≤ 89.4
VAE with analytic entropy	≤ 88.1
VAE with score function gradient	≤ 87.9
Normalizing flows [5]	≤ 85.8
Hierarchical variational model [4]	≤ 85.4
Importance-weighted auto-encoders ($K = 50$) [1]	≤ 86.3
HVM with IWAE objective ($K = 5$)	≤ 85.2
Rényi divergence ($\alpha = -1$)	≤ 140.5

Inference methods for a probabilistic decoder on binarized MNIST. The Edward PPL makes it easy to experiment with many algorithms.