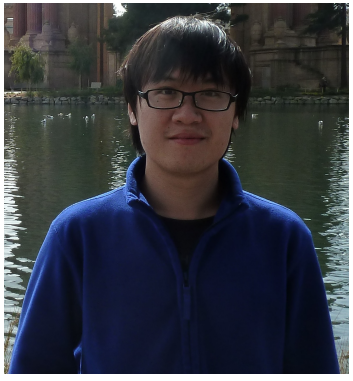
Edward: A library for probabilistic modeling, inference, and criticism

Dustin Tran
Columbia University

Alp Kucukelbir

Adji Dieng

Maja Rudolph

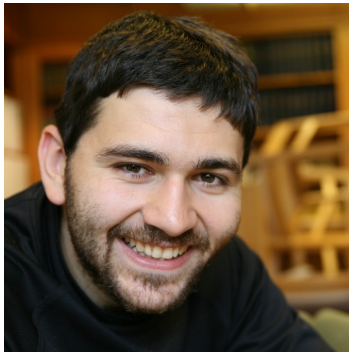


Dawen Liang

David Blei

Matt Hoffman

Kevin Murphy

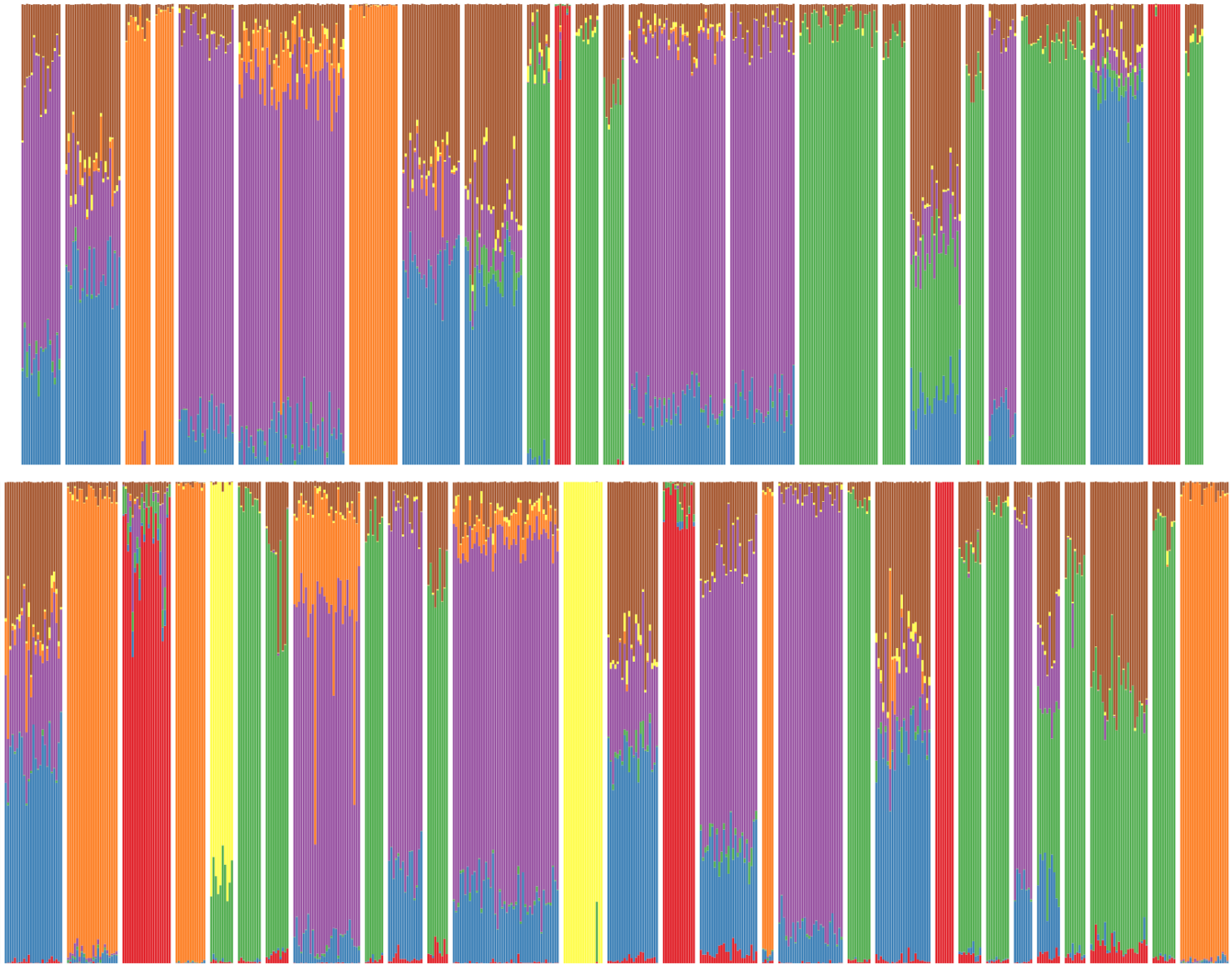


Eugene Brevdo

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 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

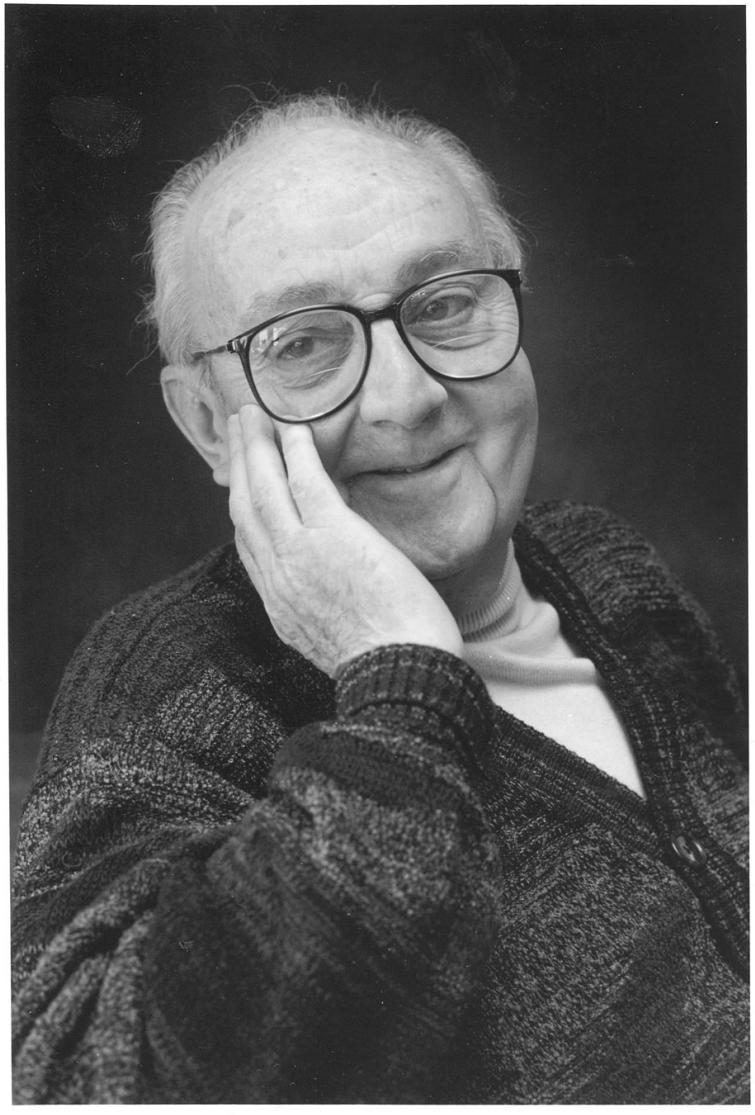
Feature extraction and normalization.

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)

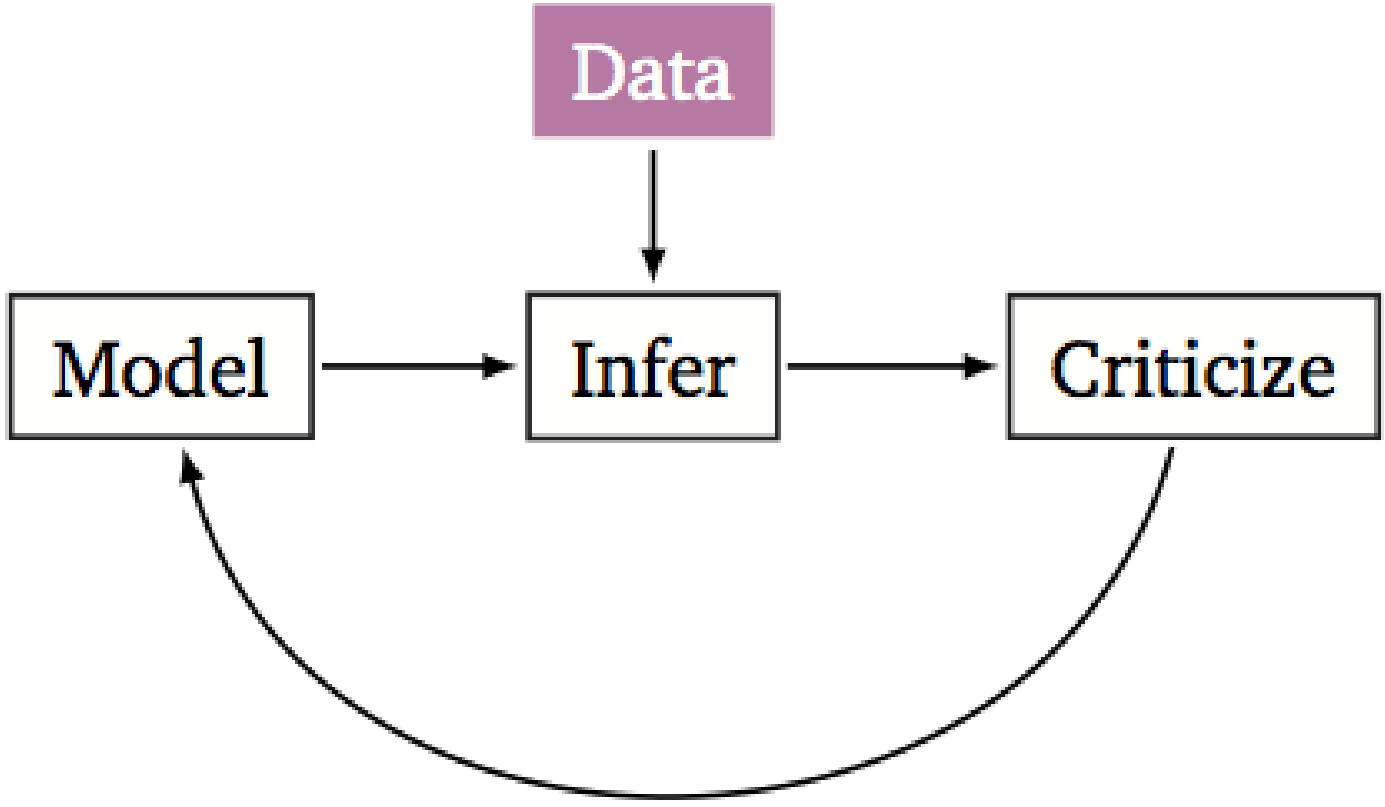


An iterative process for science:

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(tf.random_normal([])))
qp_b = tf.nn.softplus(tf.Variable(tf.random_normal([])))
qp = Beta(a=qp_a, b=qp_b)

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

# CRITICISM
x_post = ed.copy(x, {p: qp})
T = lambda xs, zs: return 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 (while still supporting PPs).

Model: Random Variables

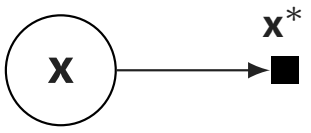
A random variable **x** is an *object* parameterized by tensors θ^*.

```
# 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)
```

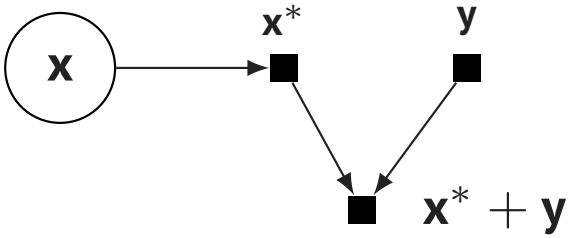
It is equipped with methods such as `log_prob()` and `sample()`.

Model: Random Variables

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

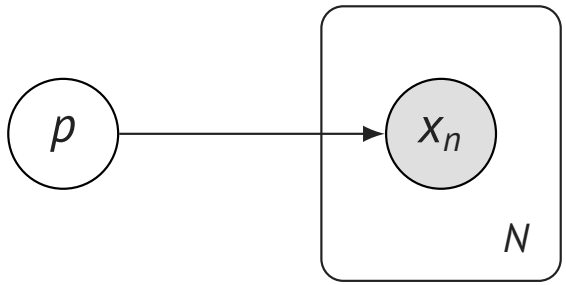


This enables ops on the computational graph. They operate on $\mathbf{x}^*$.




```
x = Normal(mu=tf.constant([0.0]*10), sigma=tf.constant([1.0]*10))
y = tf.constant(5.0)
x + y, x - y, x * y, x / y
tf.nn.tanh(x * y)
x[2] # 3rd normal rv in the vector
```

Model: Directed Graphical Models



$$p \sim \text{Beta}(1, 1)$$
$$x_n \sim \text{Bernoulli}(p)$$

To form a directed edge between random variables, $\mathbf{p} \to \mathbf{x}$, we input $\mathbf{p}$ into $\mathbf{x}$. This parameterizes $\mathbf{x}$ by $\mathbf{p}^*$, forming $p(\mathbf{x} \mid \mathbf{p}^*)$.

```
p = Beta(alpha=1.0, beta=1.0)
x = Bernoulli(p=tf.tile(p, N))
```

Model: Directed Graphical Models

The model defines a computational graph.


```
p = Beta(alpha=1.0, beta=1.0)
x = Bernoulli(p=tf.tile(p, N))
```

Model: Directed Graphical Models

The model defines a computational graph.

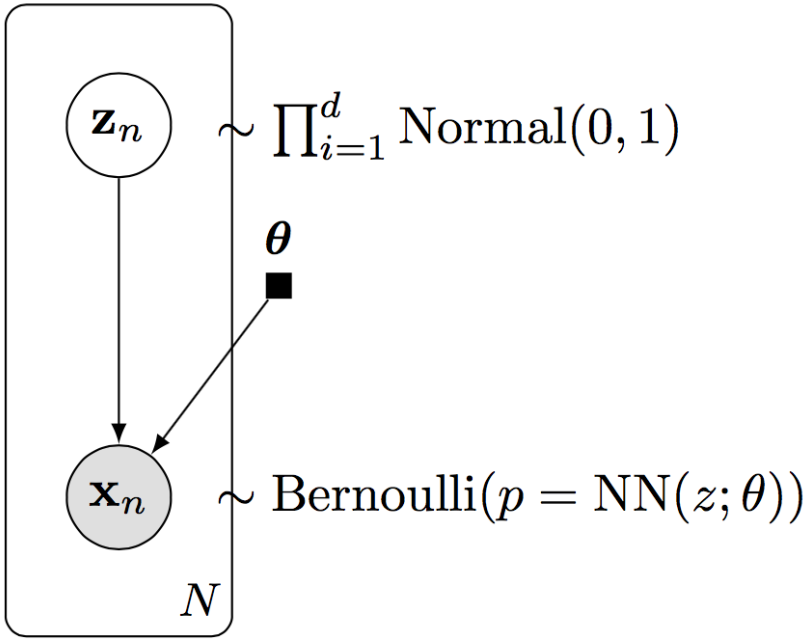

```
p = Beta(alpha=1.0, beta=1.0)
x = Bernoulli(p=tf.tile(p, N))
```

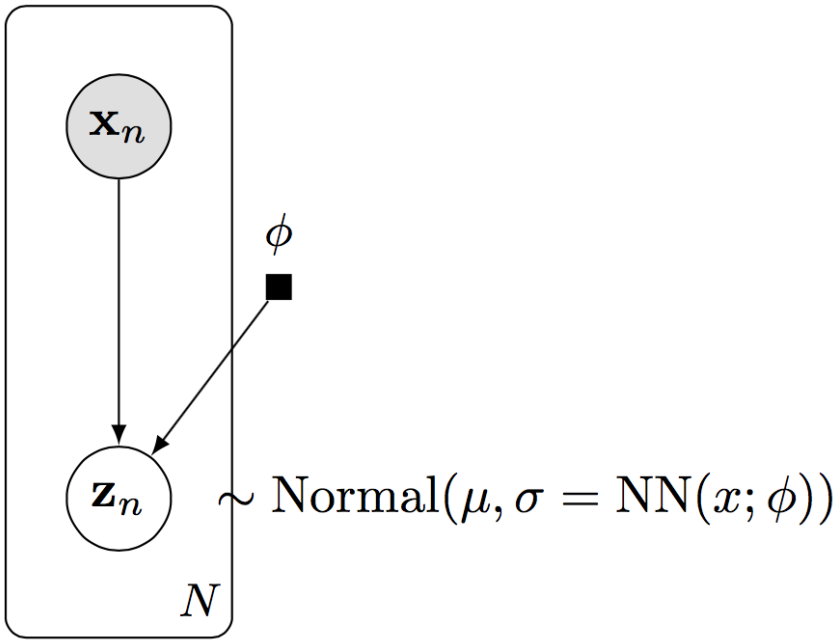
Running the graph for **x** will:

1. Generate a probability $\mathbf{p}^* \sim \text{Beta}(1, 1)$;
2. Generate data $\mathbf{x}^* \sim \prod_{n=1}^{N} \text{Bernoulli}(\mathbf{p}^*)$.

Directed structure is exposed in the computational graph. We can now write model-specific algorithms (and generic algorithms).

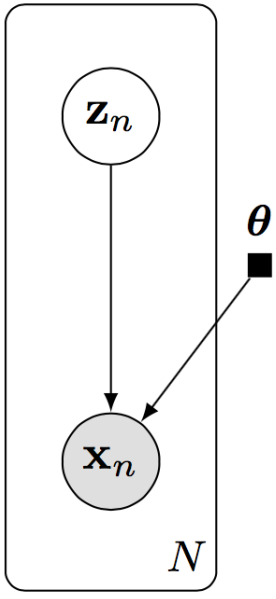
Example: Variational Auto-encoder for Binarized MNIST



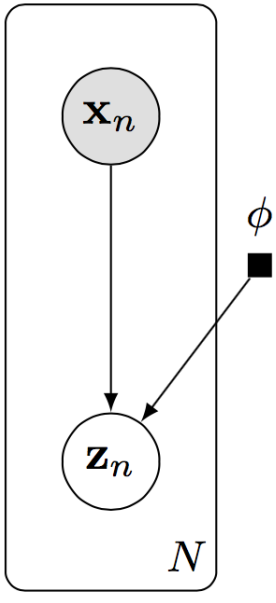


(Kingma & Welling, 2014; Rezende et al., 2014)

Example: Variational Auto-encoder for Binarized MNIST



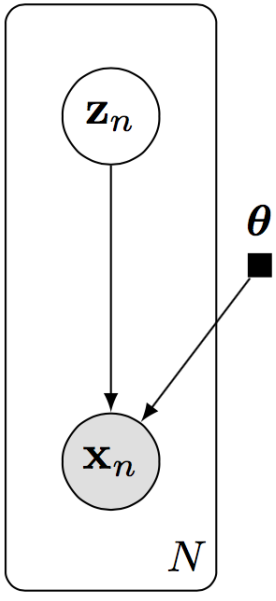

```
z = Normal(mu=tf.zeros([N, d]), sigma=tf.ones([N, d]))
hidden = Dense(64, activation=K.relu)(z.value())
x = Bernoulli(logits=Dense(28 * 28)(hidden))
```

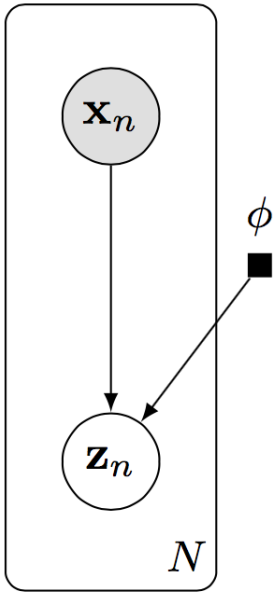



```
x_ph = ed.placeholder(tf.float32, [N, 28 * 28])
hidden = Dense(64, activation=K.relu)(x_ph)
qz = Normal(mu=Dense(d)(hidden),
            sigma=Dense(d, activation=K.softplus)(hidden))
```

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

Example: Variational Auto-encoder for Binarized MNIST



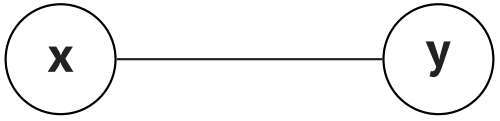



```
N = 1000
d = 10

z = Normal(mu=tf.zeros([N, d]), sigma=tf.ones([N, d]))
hidden = Dense(64, activation=K.tanh)(z)
x = Bernoulli(logits=Dense(28*28)(hidden))

x_ph = tf.placeholder(tf.float32, [N, 28*28])
hidden = Dense(64, activation=K.tanh)(x_ph)
qz = Normal(mu=Dense(d)(hidden),
            sigma=Dense(d, activation=K.softplus)(hidden))

x_data = np.loadtxt('x.txt', dtype='float32')
data = {x: x_data, x_ph: x_data}
inference = VI({z: qz}, data)
```

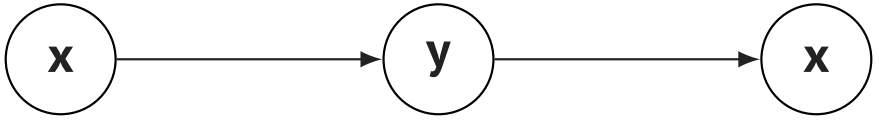
(Kingma & Welling, 2014; Rezende et al., 2014)

Model: Undirected Graphical Models

Consider forming an undirected edge between random variables, $\mathbf{x} - \mathbf{y}$.



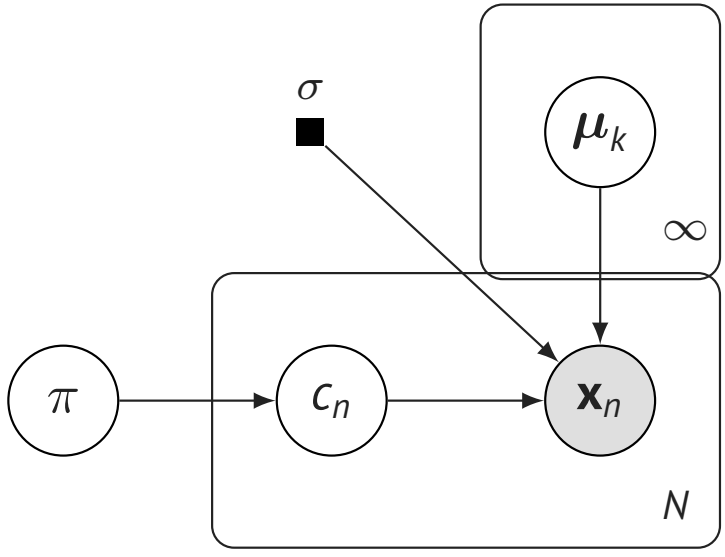
We represent the graph as a directed model and tie nodes during inference.




```
x_tied = tf.constant(0.0)
y = Normal(mu=x_tied, sigma=1.0)
x = Normal(mu=y, sigma=1.0)
```

Undirected structure is exposed in the computational graph. We can now write model-specific algorithms (and generic algorithms).

Model: BNP & Probabilistic Programs

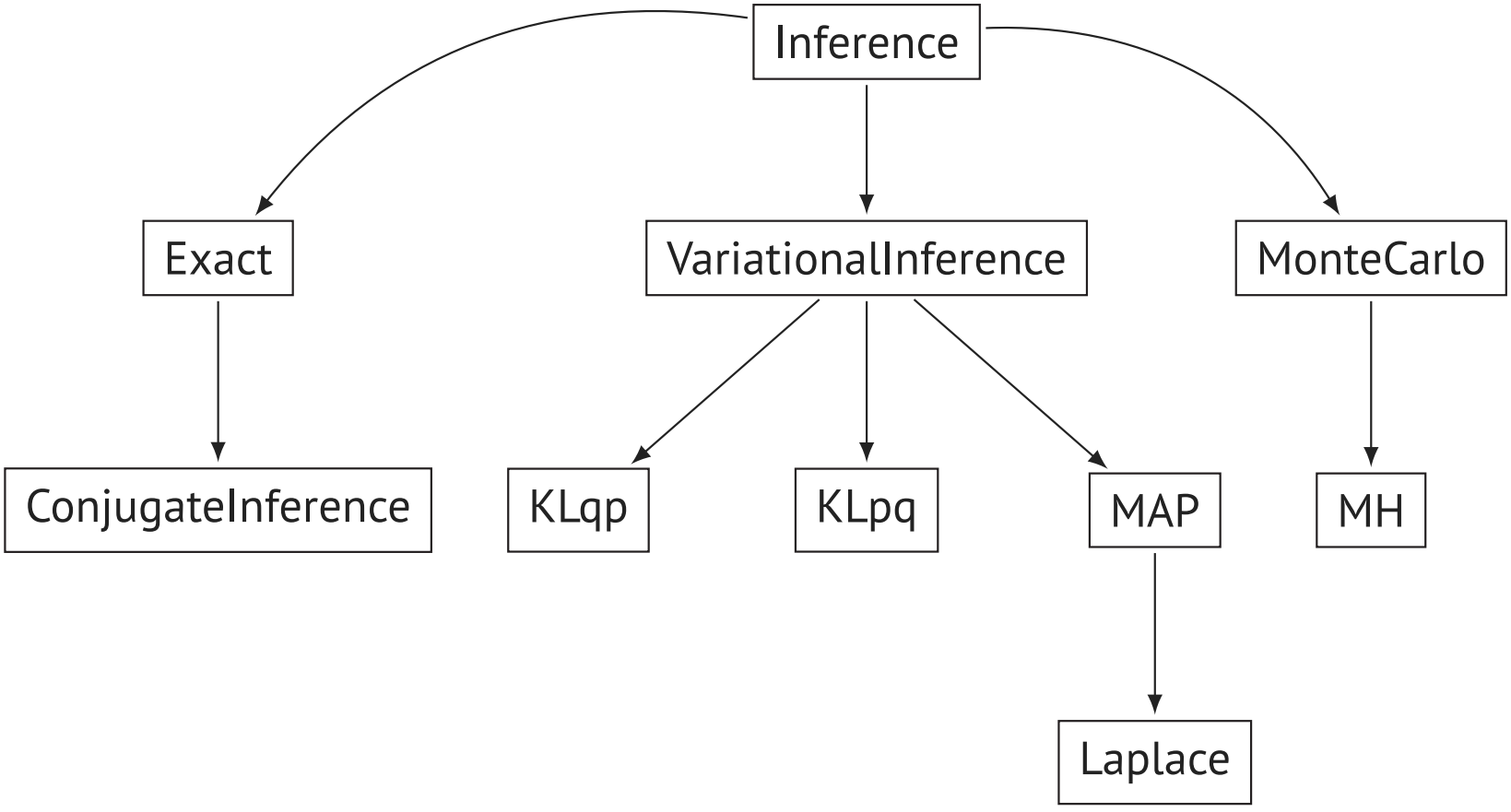


Bayesian nonparametric models also exist. They are enabled with control flow ops such as `tf.while_loop()`.

This is analogous to the implementation of dynamic RNNs.

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

Inference



An inference algorithm is a *class*. Algorithms with the same parent class share parent methods.

Inference: Background

Given

- Data set $\mathbf{x}$.
- Probability model $p(\mathbf{x}, \mathbf{z})$.

Goal

- Infer posterior $p(\mathbf{z} \mid \mathbf{x})$.

Inference: Background

Given

- Data set $\mathbf{x}$.
- Probability model $p(\mathbf{x}, \mathbf{z})$.

Goal

- Infer posterior $p(\mathbf{z} \mid \mathbf{x})$.

We take a variational perspective. All inference posits a family of distributions $q(\mathbf{z}; \boldsymbol{\lambda})$ and tries to match the posterior,

$$q(\mathbf{z}; \boldsymbol{\lambda}) \approx p(\mathbf{z} \mid \mathbf{x}).$$

Inference

The **inputs** to all algorithms are
(1) **z**, binding latent variables to variational factor;
(2) **x**, binding observed variables to observations.

```
qbeta = RandomVariable()
qz = RandomVariable()
inference = VI({beta: qbeta, z: qz}, data={x: np.array()})
```

```
qbeta = Empirical()
qz = Empirical()
inference = MCMC({beta: qbeta, z: qz}, data={x: np.array()})
```

```
qbeta = PointMass()
qz = PointMass()
inference = MAP({beta: qbeta, z: qz}, data={x: np.array()})
```

We extract pieces of the model during inference by graph copying.

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.

Thanks!

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

edwardlib.org