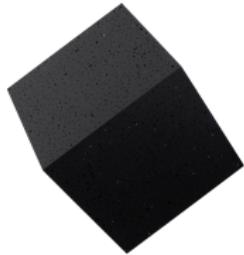


# **Edward: A library for probabilistic modeling, inference, and criticism**

Dustin Tran  
Columbia University, OpenAI





Alp Kucukelbir



Adji Dieng



Maja Rudolph



Dawen Liang



David Blei



Matt Hoffman



Kevin Murphy



Eugene Brevdo

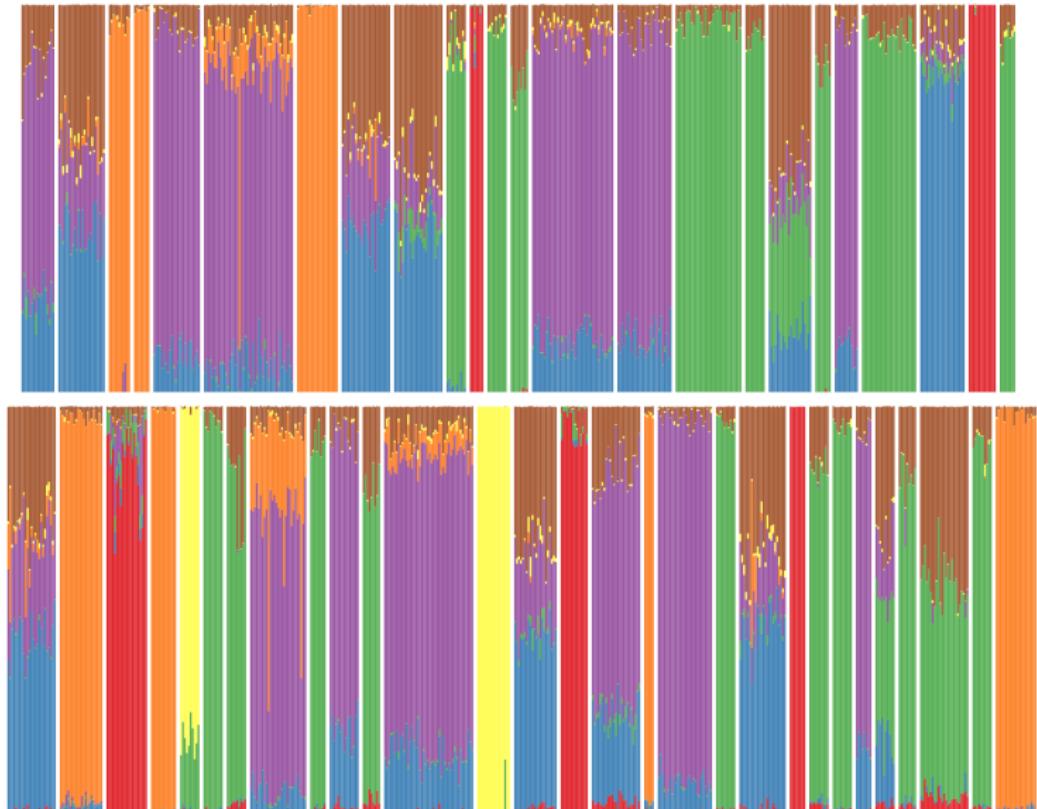


Rif Saurous

<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

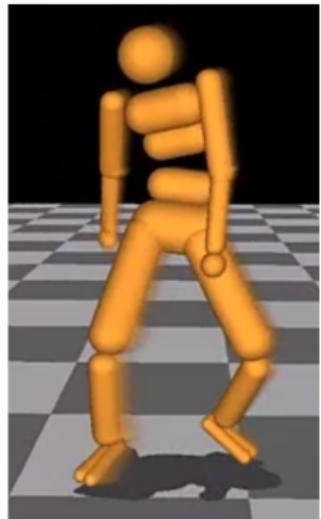
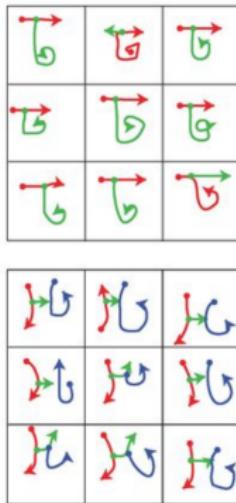
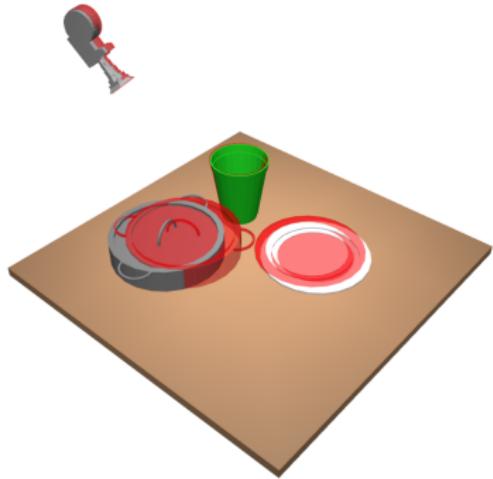
Topics found in 1.8M articles from the New York Times

[Hoffman, Blei, Wang, Paisley 2013]



Population analysis of 2 billion genetic measurements

[Gopalan+ 2017]



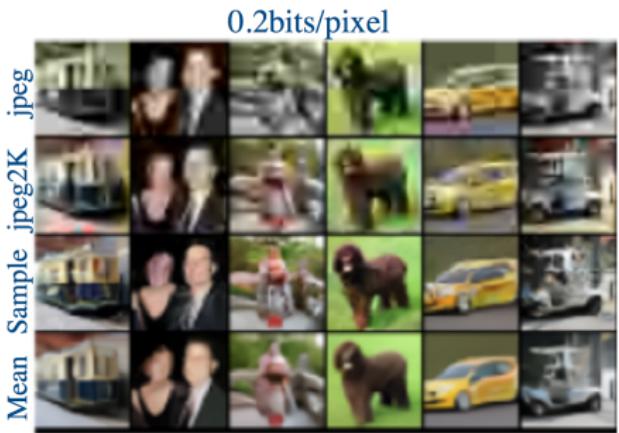
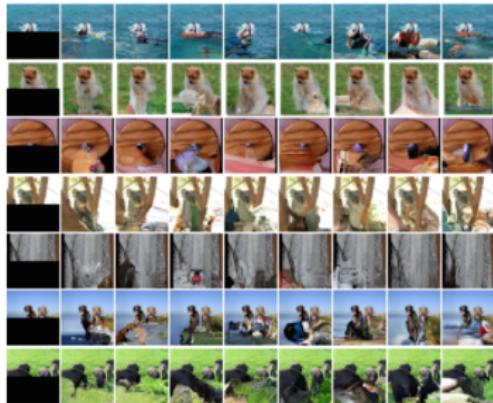
Understanding scenes, concepts, and control

[Eslami+ 2016; Lake+ 2015]



Exploratory analysis of 1.7M taxi trajectories, in Stan

[Kucukelbir+ 2017]



Compression and content generation

[Van der Oord+ 2016; Gregor+ 2016]

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

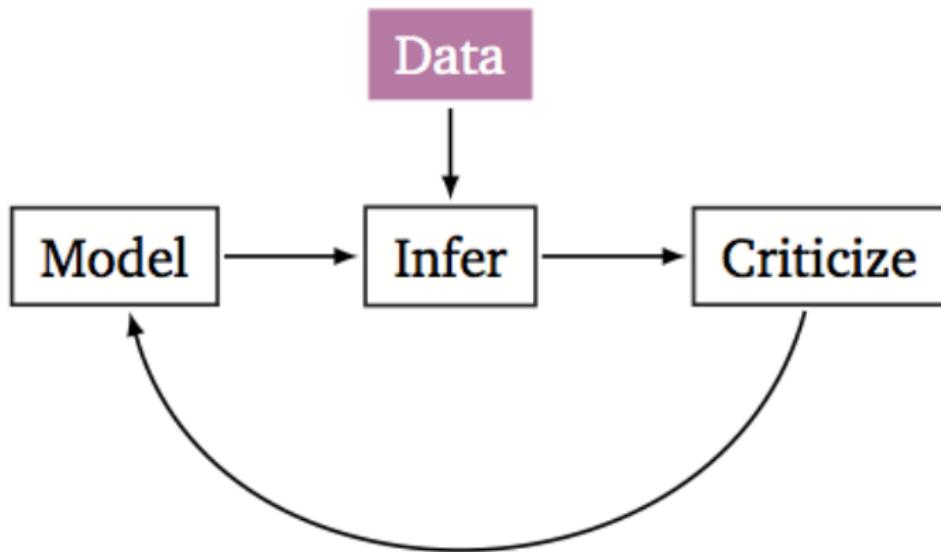


An iterative process for science:

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]

**Edward** is a probabilistic programming language built on TensorFlow.

### *Modeling*

- Composable Turing-complete language of random variables.
- Many data types, tensor vectorization, broadcasting, 3rd party support.
- Examples: Graphical models, neural networks, probabilistic programs.

### *Inference*

- Composable language for hybrids, message passing, data subsampling.
- Infrastructure to develop your own algorithms.
- Examples: Black box VI, Hamiltonian MC, stochastic gradient MCMC.

### *Criticism*

- Examples: Scoring rules, hypothesis tests, predictive checks.

Features include autodiff, multi-GPUs, distributed, XLA, quantization.

[Code](#)[Issues 99](#)[Pull requests 14](#)[Projects 1](#)[Pulse](#)[Graphs](#)

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](#)[1,680 commits](#)[17 branches](#)[24 releases](#)[45 contributors](#)Branch: [master](#) ▾[New pull request](#)[Find file](#)[Clone or download](#) ▾ [dustinvtran committed on GitHub](#) Improved training of Wasserstein GANs (#625) ...

Latest commit c92c141 a day ago

 [docker](#)

Make miscellaneous revisions to documentation (#562)

a month ago

[Sign Up](#)[Log In](#)

D

D

D

[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>

Faez 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 hundred users & many contributors.

# Who is Using Edward?

## Users

1. Machine learning enthusiasts, data scientists, business analysts  
*(ex. hierarchical GLMs, mixture models, MAP, MCMC, ...)*
2. Probabilistic graphical modeling community  
*(ex. latent Dirichlet allocation, variational inference, Gibbs)*
3. Bayesian deep learning community  
*(ex. deep generative models, Bayesian NNs, black box inference)*

## Developers

1. David Blei's group
2. Google Brain (*design*)
3. Matt Hoffman (*conjugacy*), Emily Fox's group (*time series + SGMCMC*),  
Justin Bayer (*stochastic RNNs*), John Pearson (*neuroscience*), a few  
Master's/Ph.D. students.
4. Collaboration continues to evolve. Contact us! (+visit the Forum)

# How do we use Edward?

[Demo]

[edwardlib.org/getting-started](http://edwardlib.org/getting-started)

# Model

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

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

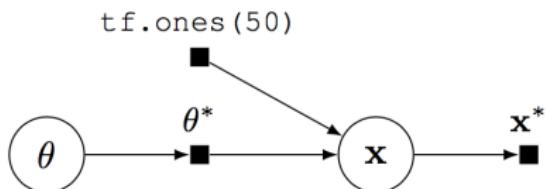
## Example: Beta-Bernoulli

Consider a Beta-Bernoulli model,

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

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

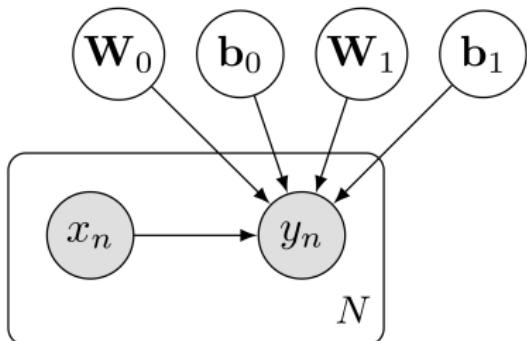
```
1 theta = Beta(1.0, 1.0)
2 x = Bernoulli(probs=tf.ones(50) * theta)
```



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

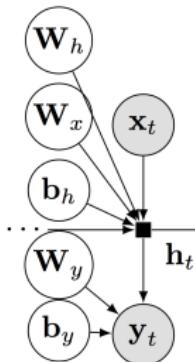
All computation is represented on the graph, enabling us to leverage model structure during inference.

## Example: Bayesian neural network for classification



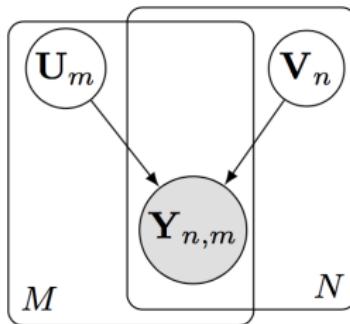
```
1 W_0 = Normal(mu=tf.zeros([D, H]), sigma=tf.ones([D, H]))
2 W_1 = Normal(mu=tf.zeros([H, 1]), sigma=tf.ones([H, 1]))
3 b_0 = Normal(mu=tf.zeros(H), sigma=tf.ones(L))
4 b_1 = Normal(mu=tf.zeros(1), sigma=tf.ones(1))
5
6 x = tf.placeholder(tf.float32, [N, D])
7 y = Bernoulli(logits=tf.matmul(tf.nn.tanh(tf.matmul(x,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

# Example: 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)
```

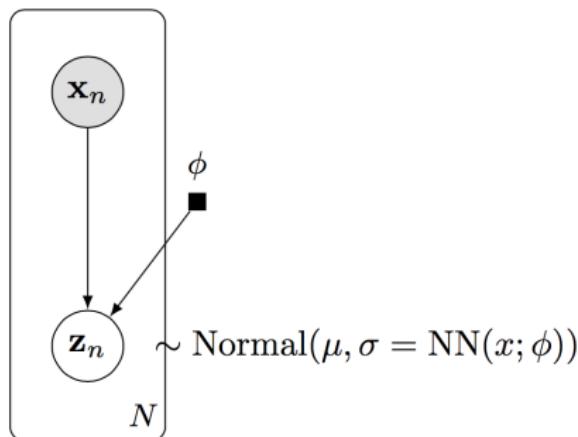
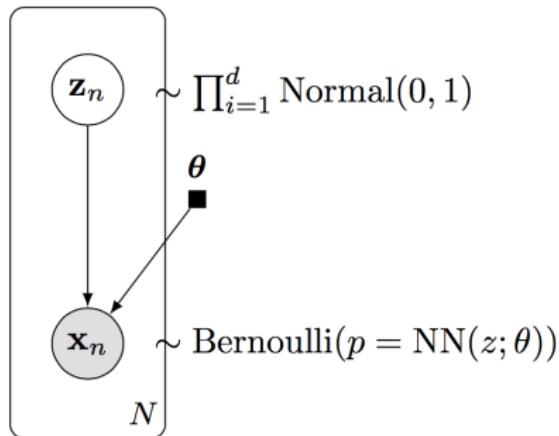
# Example: Gaussian Matrix Factorization



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

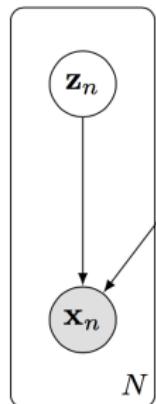
[Bishop & Tipping 1997; Salakhutdinov & Mnih 2007]

## Example: Variational Auto-Encoder for Binarized MNIST

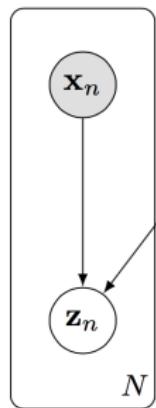


[Kingma & Welling 2014; Rezende+ 2014]

# 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))
```

## Example: Variational Auto-Encoder for Binarized MNIST

[Demo]

[github.com/blei-lab/edward/blob/master/  
examples/vae.py](https://github.com/blei-lab/edward/blob/master/examples/vae.py)

# 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  $\mathbf{z}, \beta$ , 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.

# Inference

Variational inference. It uses a variational model.

```
1 qbeta = Normal(loc=tf.Variable(tf.zeros([K, D])),  
2                  scale=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. It uses an Empirical approximation.

```
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})
```

Conjugacy & exact inference. It uses symbolic algebra on the graph.

# 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
<b>Edward (12 CPU)</b>	<b>8.2</b>
Handwritten TensorFlow (GPU)	5.0
<b>Edward (GPU)</b>	<b>4.9</b> (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: Recent Methods in Variational Inference

Inference method	Neg. log-likelihood
VAE [Kingma & Welling 2014]	$\leq 88.2$
VAE without analytic KL	$\leq 89.4$
VAE with analytic entropy	$\leq 88.1$
VAE with score function gradient	$\leq 87.9$
Normalizing flows [Rezende & Mohamed 2015]	$\leq 85.8$
Hierarchical variational model [Ranganath+ 2016]	$\leq 85.4$
Importance-weighted auto-encoders ( $K = 50$ ) [Burda+ 2016]	$\leq 86.3$
HVM with IWAE objective ( $K = 5$ )	$\leq 85.2$
Rényi divergence ( $\alpha = -1$ ) [Li & Turner 2016]	$\leq 140.5$

**Edward enables fast experimentation with state-of-the-art methods.**

# Summary

1. Edward is a library 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, bridging research design and deployment.

## Current directions

We are robustifying Edward.

1. Bug fixes, improved vectorization/types, errors/warnings.
2. More built-in algorithms and model / inference tutorials.
3. Benchmarks across many modeling & inference applications.

We are applying Edward for AI and scientific research.

1. Generative models for language and vision. (OpenAI)
2. Implicit generative models and variational inference. (OpenAI)
3. Causal models for genome wide association studies. (Storey, Engelhardt)

# References



[edwardlib.org](http://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, 2016.
- **D. Tran**, M.D. Hoffman, R.A. Saurous, E. Brevdo, K. Murphy, and D.M. Blei.  
Deep probabilistic programming.  
International Conference on Learning Representations, 2017.