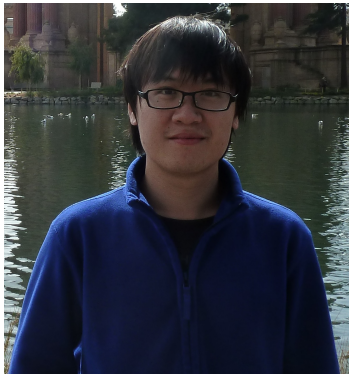
Edward: A library for probabilistic modeling, inference, and criticism

Dustin Tran
Columbia University, OpenAI

Alp Kucukelbir

Adji Dieng

Maja Rudolph

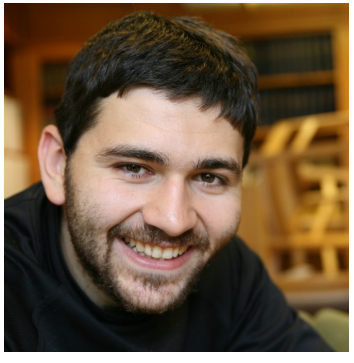


Dawen Liang

David Blei

Matt Hoffman

Kevin Murphy

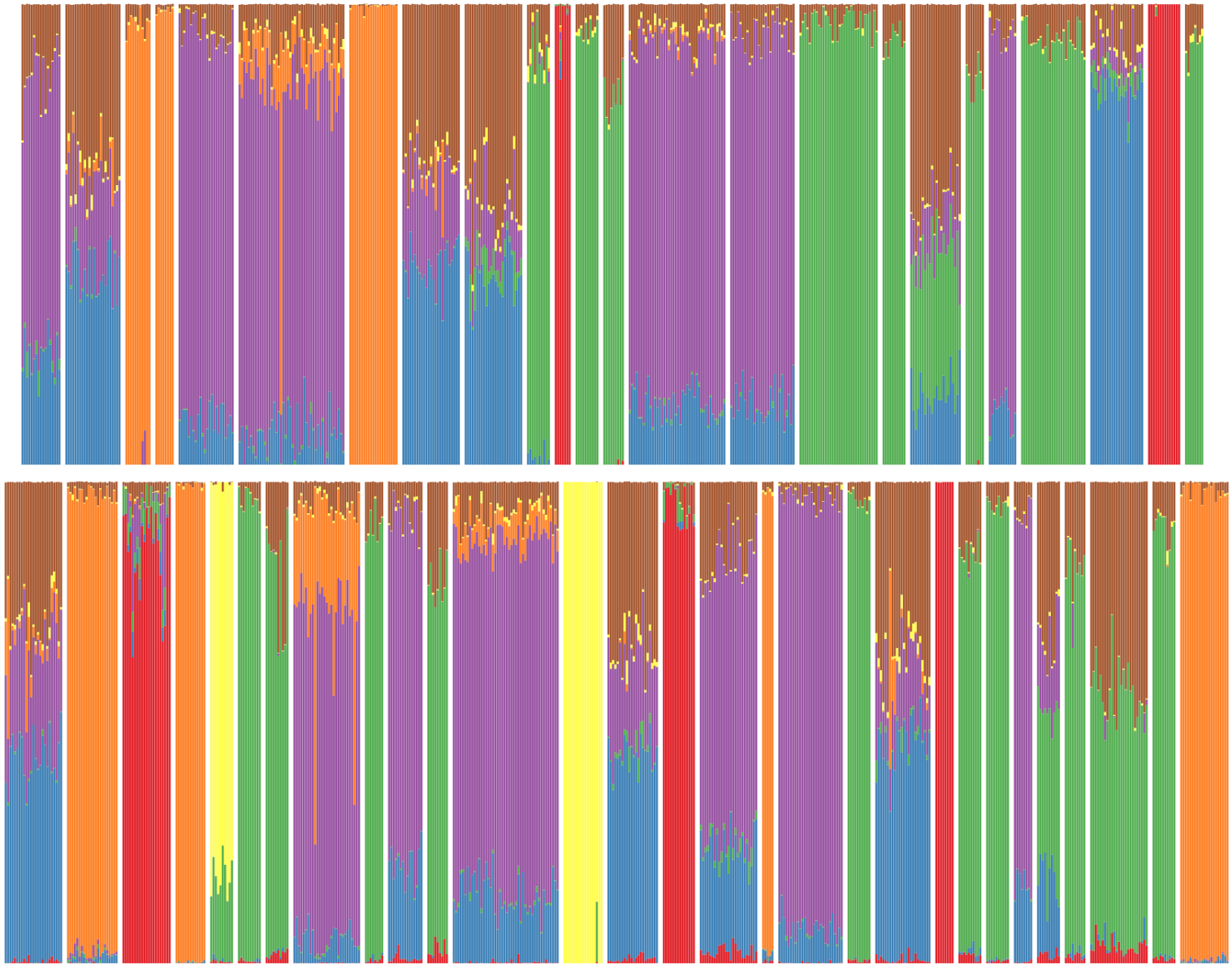


Eugene Brevdo

Rif Saurous

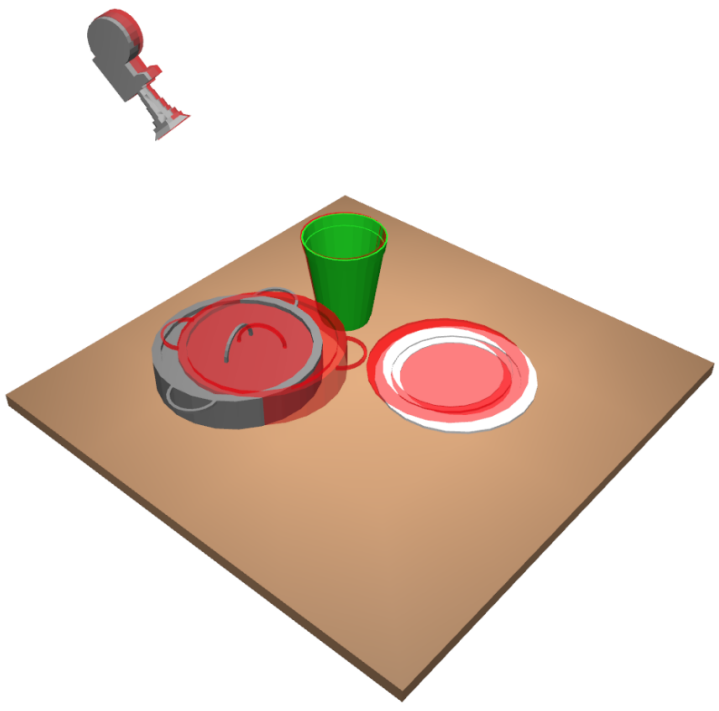
Topics found in 1.8M articles from the New York Times

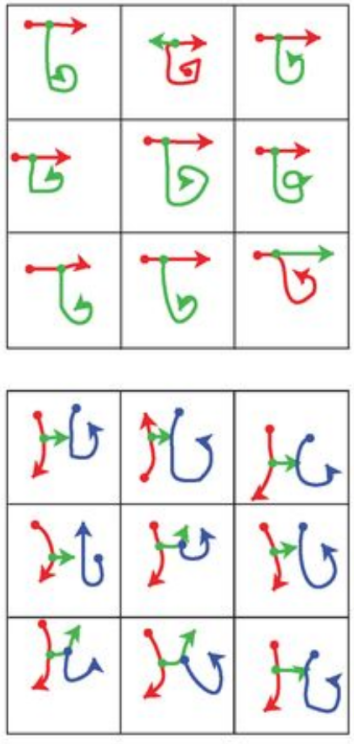
[Hoffman, Blei, Wang, Paisley 2013]

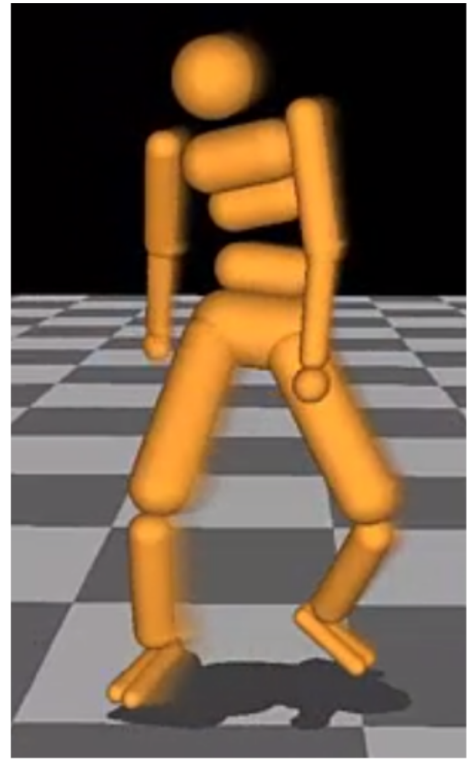


Population analysis of 2 billion genetic measurements

[Gopalan+ 2017]





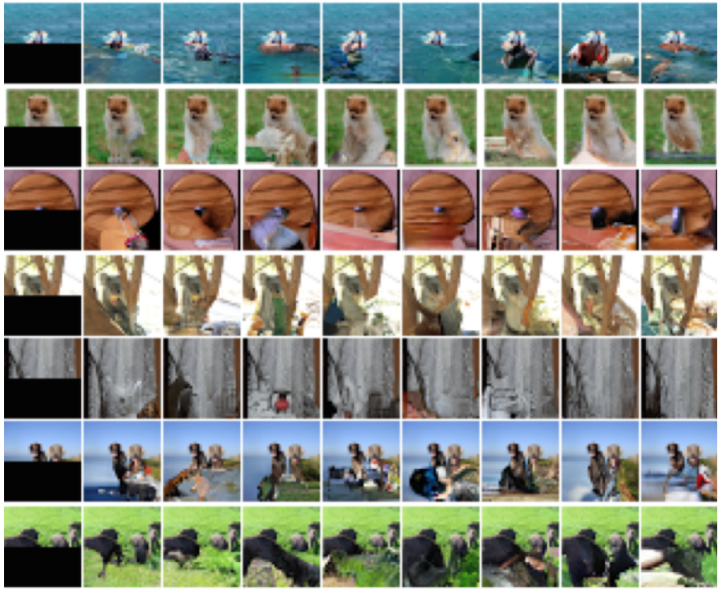


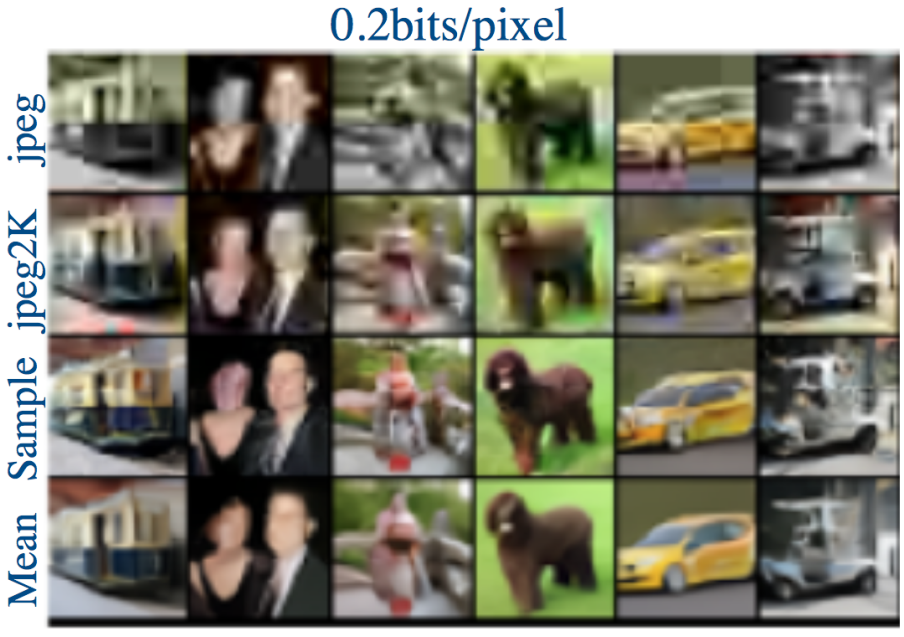
Understanding scenes, concepts, and control

[Eslami+ 2016; Lake+ 2015]

Exploratory analysis of 1.7M taxi trajectories, in Stan

[Kucukelbir+ 2017]

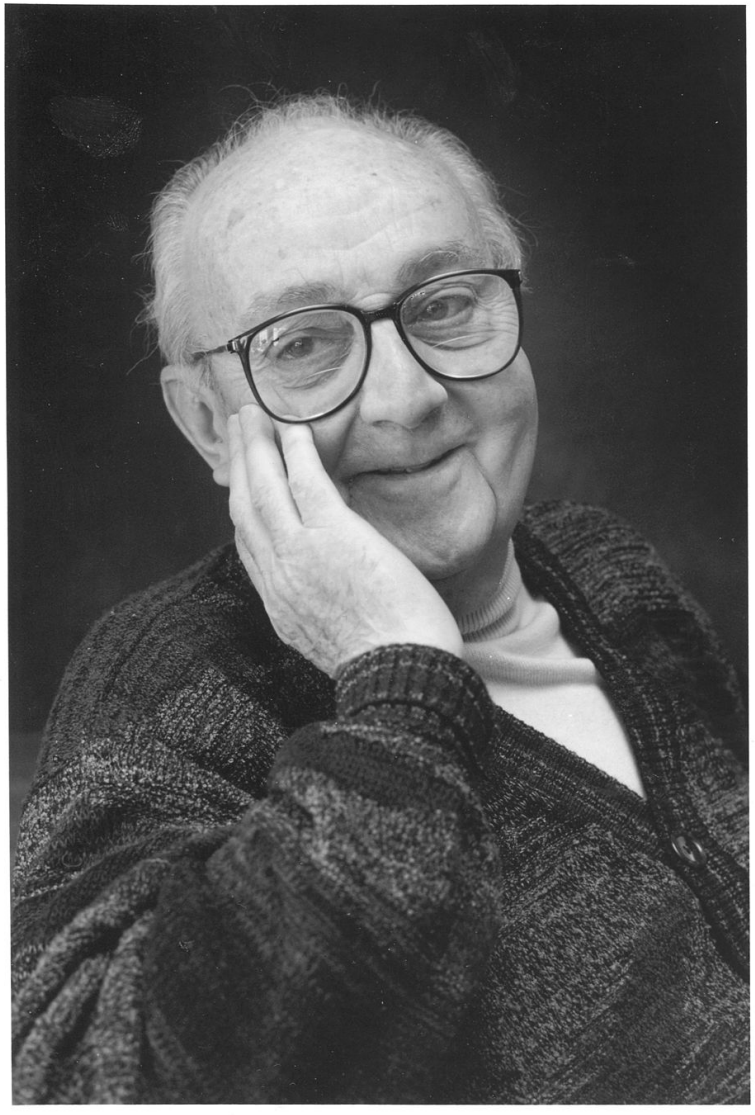




Compression and content generation

[Van der Oord+ 2016; Gregor+ 2016]

George E.P. Box (1919 - 2013)

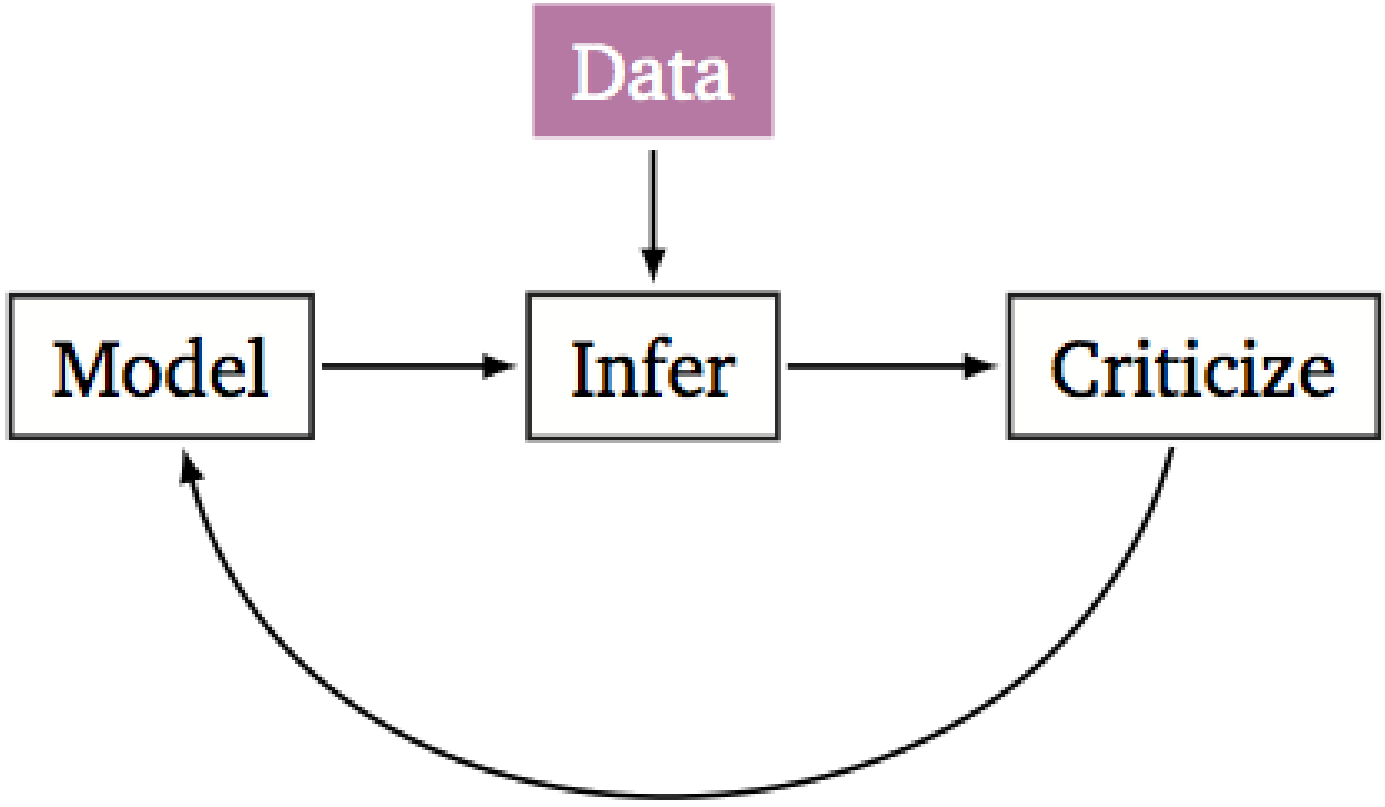


An iterative process for science:

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

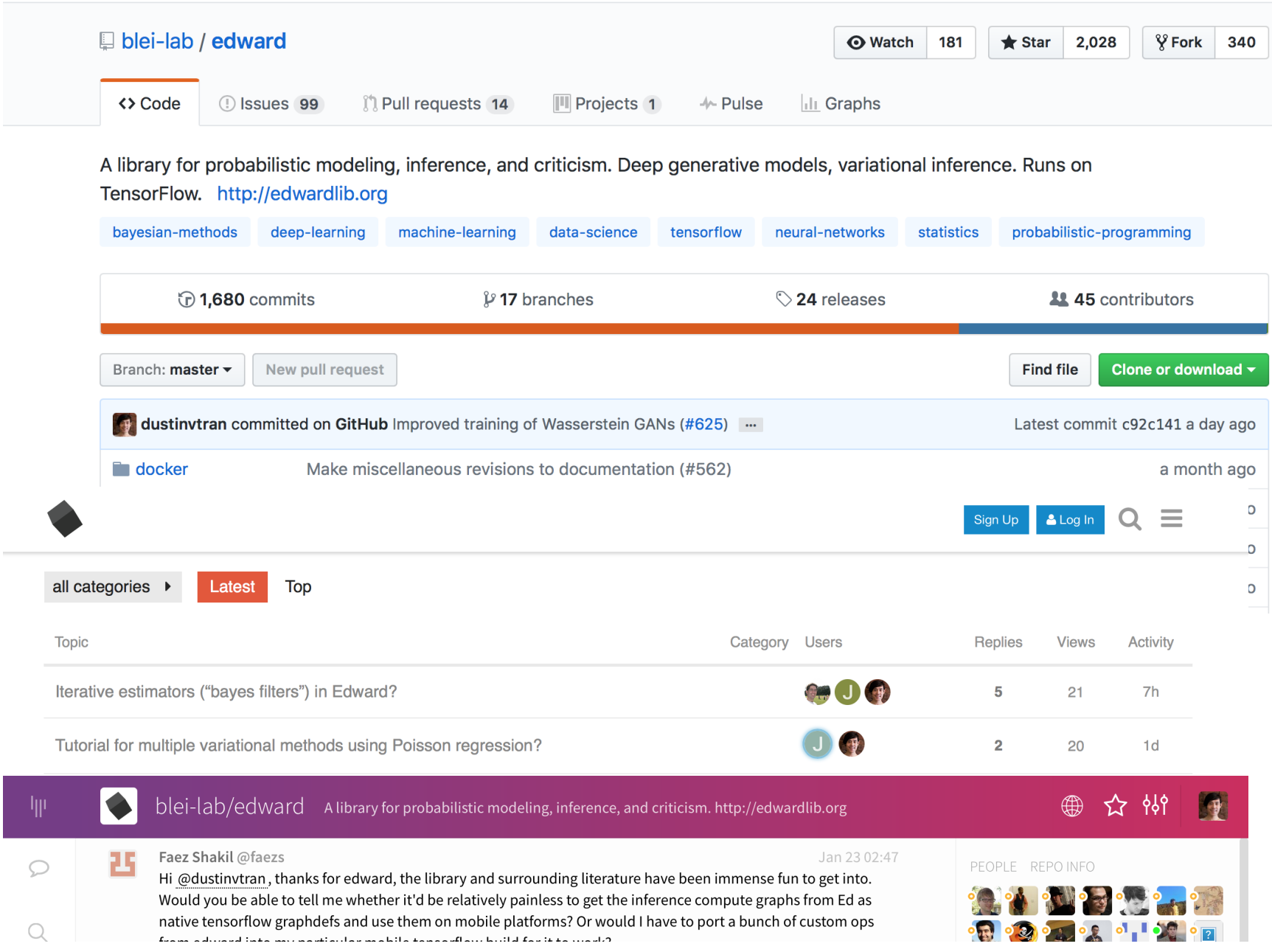
- Composable language for hybrids, message passing, data subsampling.
- Infrastructure to develop your own algorithms.
- Examples: Black box VI, Hamiltonian MC, stochastic gradient MCMC.

Criticism

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

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

[Tran+ 2016, 2017]



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

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

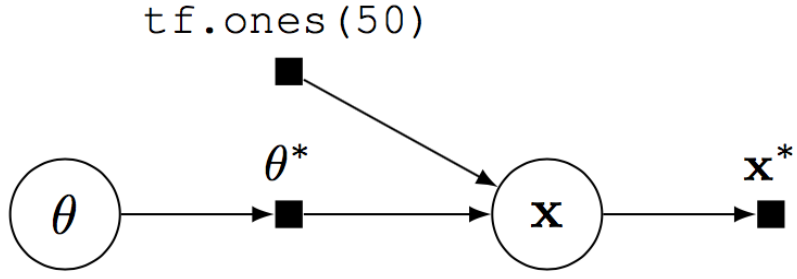
Example: Beta-Bernoulli

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

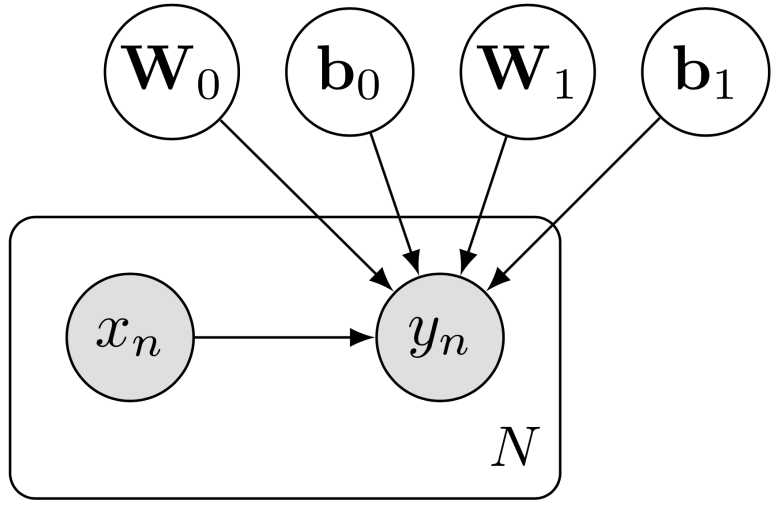
```
1  theta = Beta(1.0, 1.0)
2  x = Bernoulli(probs=tf.ones(50) * theta)
```



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

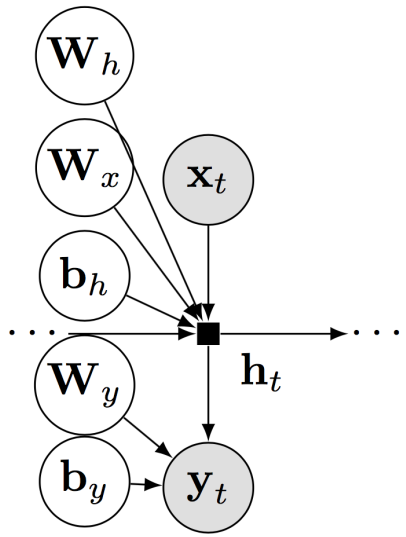
All computation is represented on the graph, enabling us to leverage model structure during inference.

Example: Bayesian neural network for classification



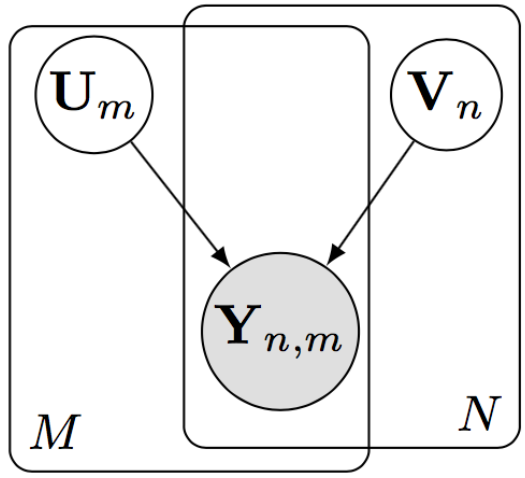

```
W_0 = Normal(mu=tf.zeros([D, H]), sigma=tf.ones([D, H]))
W_1 = Normal(mu=tf.zeros([H, 1]), sigma=tf.ones([H, 1]))
b_0 = Normal(mu=tf.zeros(H), sigma=tf.ones(L))
b_1 = Normal(mu=tf.zeros(1), sigma=tf.ones(1))

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: 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)
```

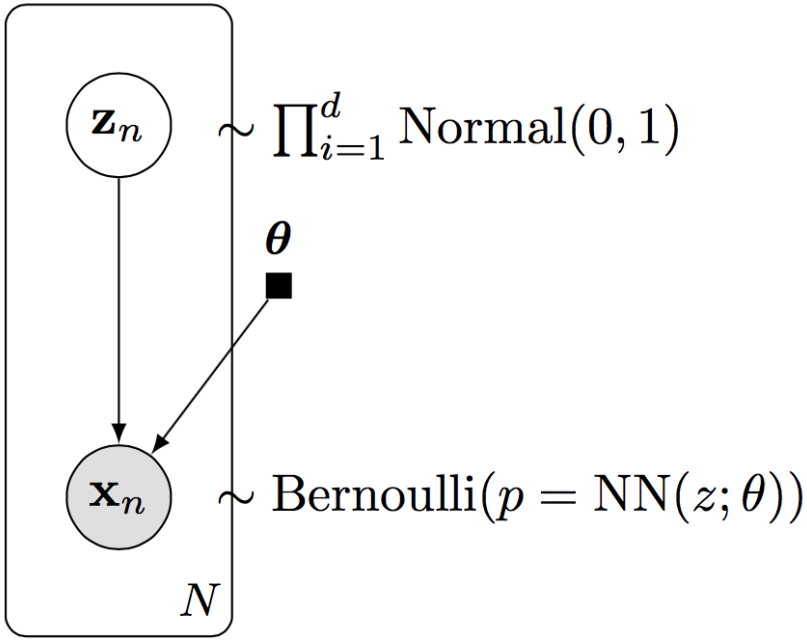
Example: Gaussian Matrix Factorization

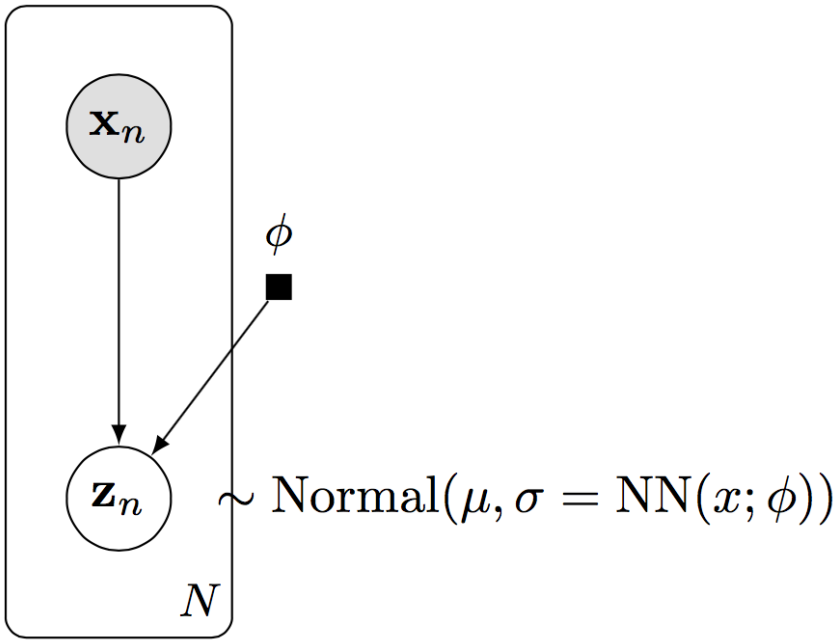



```
N = 10
M = 10
K = 5  # latent dimension

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

[Bishop & Tipping 1997; Salakhutdinov & Mnih 2007]

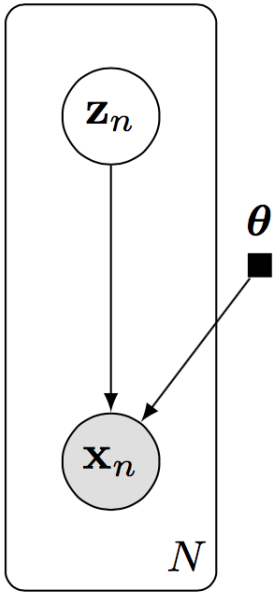
Example: Variational Auto-Encoder for Binarized MNIST



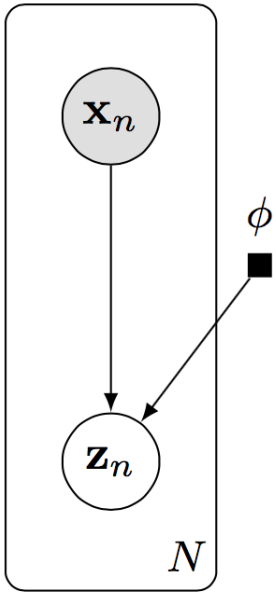


[Kingma & Welling 2014; Rezende+ 2014]

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))
```

[Kingma & Welling 2014; Rezende+ 2014]

Example: Variational Auto-Encoder for Binarized MNIST

[Demo]

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 **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.

Inference

Variational inference. It uses a variational model.

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

```
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})
```

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

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: Recent Methods in Variational Inference

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

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

[Tran+ 2017]

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

- **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.