

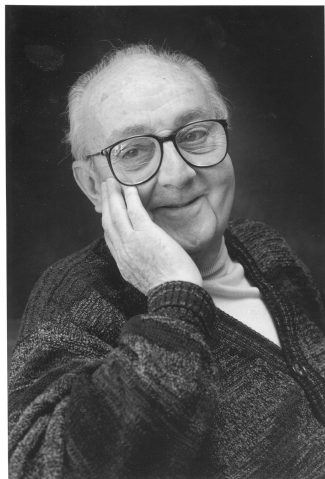
Probabilistic programming with Edward

John Reid

Biostatistics Unit,
School of Clinical Medicine,
Cambridge University

April 25, 2017

George E.P. Box (1919 - 2013)

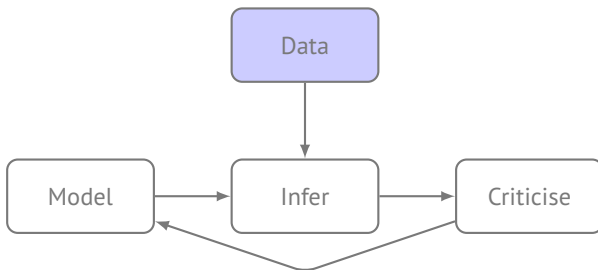


An iterative process for science:

1. Build a model of the science
2. Infer the model given data
3. Criticize the model given data

(Box and Hunter, 1962, 1965; Box and Hill, 1967; Box, 1976, 1980)

Box's Loop



Edward is a library designed around this loop
(Box, 1976, 1980; David M. Blei, 2014)

Model comparisons

Inference method	Negative 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 et al., 2016b)	≤ 85.4
Importance-weighted auto-encoders ($K = 50$) (Burda et al., 2016)	≤ 86.3
HVM with IWAE objective ($K = 5$)	≤ 85.2
Rényi divergence ($\alpha = -1$) (Li & Turner, 2016)	≤ 140.5

Table 1: Inference methods for a probabilistic decoder on binarized MNIST. The Edward PPL makes it easy to experiment with many algorithms.

Edward is a probabilistic programming language, designed for fast experimentation and research (Tran et al., 2017).

Modelling

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

Inference

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

Criticism

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

Built on TensorFlow (features distributed computing, GPUs, autodiff)

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)
```

VARIATIONAL DISTRIBUTION

```
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

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

```
inference.run(n_iter=500)
```

CRITICISM

```
x_post = ed.copy(x, {p : qp})
```

```
def T(xs, zs):
```

```
    return tf.reduce_mean(xs[x_post])
```

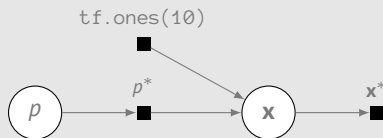
```
ed.ppc(T, data={x_post: x_data})
```

Model code

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

The random variables p and x are represented by tensors p^* and \mathbf{x}^* in the tensorflow computational graph

Computational graph



Random variables are equipped with methods for likelihoods $\log(x|p)$, expectations $\mathbb{E}_{p(x|p)}[x]$, and sampling $\sim p(x|p)$.

Graph can be executed by `x.value()` which returns the tensor \mathbf{x}^* and simulates the generative process.

Model construction

Key concept is compositionality:

- ▶ Computational graphs can contain arbitrary tensorflow constructs
- ▶ Tensorflow conditional evaluations permit nonparametric processes
- ▶ Interface with third party tensorflow libraries, e.g. Keras for deep learning

Generative model

```
from edward.models import Bernoulli, Normal
from keras.layers import Dense

z = Normal(mu=tf.zeros([N, d]), sigma=tf.ones([N, d]))
h = Dense(256, activation='relu')(z.value())
x = Bernoulli(logits=Dense(28 * 28)(h))
```


Inference abstraction

Edward's random variables can represent probabilistic models as computational graphs.

How to perform inference? We desire:

- ▶ Support for many inference classes
- ▶ The posterior can be further composed as part of a larger model

Edward abstracts this as an optimisation problem

$$\min_{\lambda, \theta} \mathcal{L}(p(\mathbf{z} \mid \mathbf{x}; \theta), q(\mathbf{z}; \phi))$$

where q can be a variational distribution, point estimate or collection of samples.

The loss for the optimisation problem is encoded in the same computational graph as the model.

MNIST variational auto-encoder

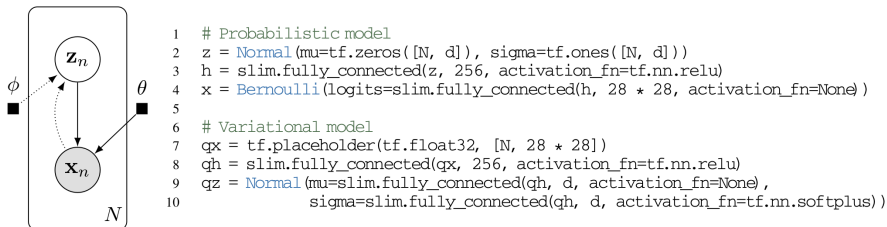


Figure 2: Variational auto-encoder for a data set of 28×28 pixel images: (left) graphical model, with dotted lines for the inference model; (right) probabilistic program, with 2-layer neural networks.

Inference

```
inference = ed.KLqp({z: qz}, data={x: qx})
inference.initialize()

for epoch in range(n_epoch):
    for t in range(n_iter_per_epoch):
        inference.update(feed_dict={x_ph: mnist.train.next_batch(M)})
```

Semi-supervised learning



Model M2 from Kingma et al., 2014

$$y \sim \text{Cat}(y|\pi)$$

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, I)$$

$$\mathbf{x}|y, \mathbf{z} \sim f(\mathbf{x}; y, \mathbf{z}, \theta)$$

55,000 binarized images, \mathbf{x}
3,000 have associated labels, y
 f is a deconvolutional network
with a Bernoulli likelihood

Variational distribution

$$q_{\phi}(y|\mathbf{x}) = \text{Cat}(\pi_{\phi}(\mathbf{x}))$$

$$q_{\phi}(\mathbf{z}|y, \mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_{\phi}(y, \mathbf{x}), \text{diag}(\boldsymbol{\sigma}_{\phi}^2(\mathbf{x})))$$

Semi-supervised learning

Loss for labelled samples

$$\mathcal{L}(\mathbf{x}, y) = -\mathbb{E}_{q_{\phi}(\mathbf{z}|y, \mathbf{x})} [\log p_{\theta}(\mathbf{x}|y, \mathbf{z}) + \log p_{\theta}(y) + \log p(\mathbf{z}) - \log q_{\phi}(\mathbf{z}|y, \mathbf{x})]$$

Loss for unlabelled samples

$$\mathcal{U}(\mathbf{x}, y) = \sum_y q_{\phi}(y|\mathbf{x}) \mathcal{L}(\mathbf{x}, y) - \mathcal{H}(q_{\phi}(y|\mathbf{x}))$$

KL-divergence and entropy are analytic

Reparameterisation trick

$$\nabla_{\{\theta, \phi\}} \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z})] = \mathbb{E}_{\mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})} [\nabla_{\{\theta, \phi\}} \log p_{\theta}(\mathbf{x}|\boldsymbol{\mu}_{\phi}(\mathbf{x}) + \boldsymbol{\sigma}_{\phi}(\mathbf{x}) \odot \boldsymbol{\epsilon})]$$

Semi-supervised learning

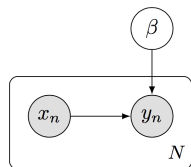


\mathbf{x} sampled from generative model

Each row has fixed y

\mathbf{z} varies across columns

GPU-accelerated Hamiltonian Monte Carlo



```
1  # Model
2  x = tf.Variable(x_data, trainable=False)
3  beta = Normal(mu=tf.zeros(D), sigma=tf.ones(D))
4  y = Bernoulli(logits=tf.dot(x, beta))
5
6  # Inference
7  qbeta = Empirical(params=tf.Variable(tf.zeros([T, D])))
8  inference = ed.HMC({beta: qbeta}, data={y: y_data})
9  inference.run(step_size=0.5 / N, n_steps=100)
```

Bayesian logistic regression for the Covertypes dataset ($N = 581012$, $D = 54$)
12-core Intel i7-5930K CPU at 3.50GHz and a NVIDIA Titan X (Maxwell) GPU
100 iterations of HMC

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

(Carpenter et al., 2017; Salvatier, Wiecki, and Fonnesbeck, 2015)

- [1] G. E. P. Box and William G. Hunter. “A Useful Method for Model-Building”. In: *Technometrics* 4.3 (1962), pp. 301–318. JSTOR:1266570.
- [2] G. E. P. Box and William G. Hunter. “The Experimental Study of Physical Mechanisms”. In: *Technometrics* 7.1 (1965), pp. 23–42. JSTOR:1266125.
- [3] G. E. P. Box and W. J. Hill. “Discrimination among Mechanistic Models”. In: *Technometrics* 9.1 (1967), pp. 57–71. JSTOR:1266318.
- [4] George E. P. Box. “Science and Statistics”. In: *Journal of the American Statistical Association* 71.356 (1976), pp. 791–799. JSTOR:2286841.
- [5] George E. P. Box. “Sampling and Bayes’ Inference in Scientific Modelling and Robustness”. In: *Journal of the Royal Statistical Society. Series A (General)* 143.4 (1980), pp. 383–430. JSTOR:2982063.
- [6] David M. Blei. “Build, Compute, Critique, Repeat: Data Analysis with Latent Variable Models”. In: *Annual Review of Statistics and Its Application* 1.1 (2014), pp. 203–232.
- [7] Dustin Tran et al. “Deep Probabilistic Programming”. In: (Jan. 13, 2017). arXiv:1701.03757 [cs, stat].
- [8] Diederik P. Kingma et al. “Semi-Supervised Learning with Deep Generative Models”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 3581–3589.
- [9] Bob Carpenter et al. “Stan : A Probabilistic Programming Language”. In: *Journal of Statistical Software* 76.1 (2017).
- [10] John Salvatier, Thomas Wiecki, and Christopher Fonnesbeck. “Probabilistic Programming in Python Using PyMC”. In: (July 29, 2015). arXiv:1507.08050 [stat].