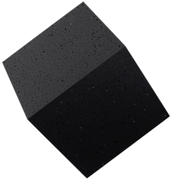
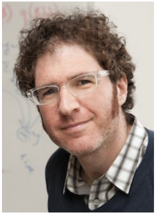
Developments in variational inference: Tradeoffs and Edward



Dustin Tran
Columbia University


Rajesh Ranganath

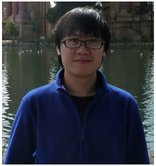

Jaan Altosaar


David Blei


Alp Kucukelbir


Adji Dieng


Maja Rudolph

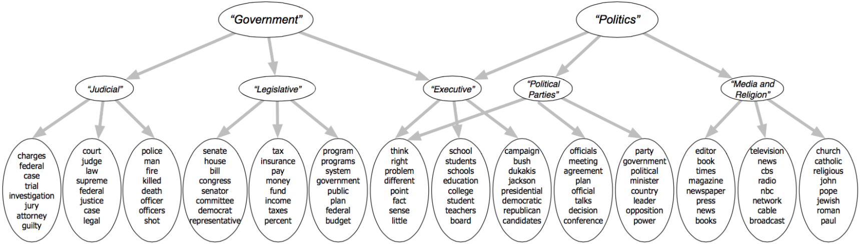

Dawen Liang


Matt Hoffman


Kevin Murphy

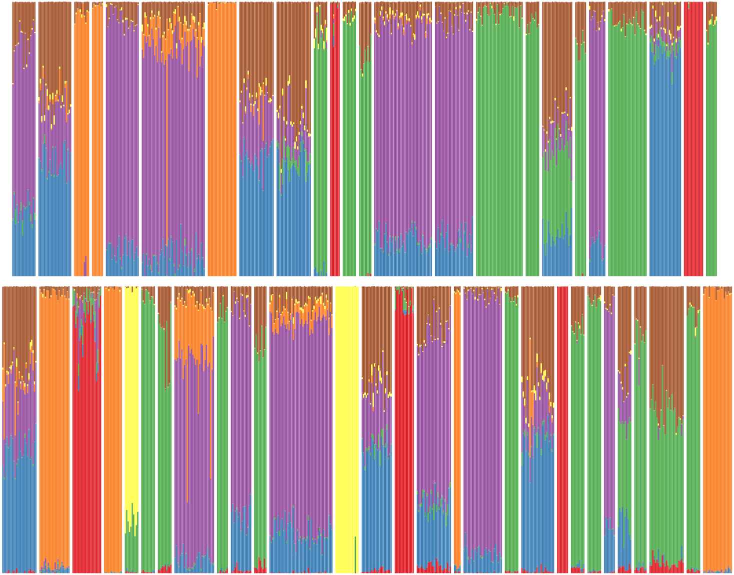

Eugene Brevdo


Rif Saurous



Hierarchy of topics found in 166K articles from the New York Times

(Ranganath et al. 2015)



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 Bayesian Inference

1. **Tradeoffs.** How do we formalize statistical and computational tradeoffs for inference?
2. **Software.** How do we design efficient and flexible software for generative models?

Background

Given

- Data set $\mathbf{x}$.
- Generative model $p(\mathbf{x}, \mathbf{z})$ with latent variables $\mathbf{z} \in \mathbb{R}^d$.

Goal

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

This is the key problem in Bayesian inference.

Background

Variational inference

- Posit a family of distributions $q \in \mathcal{Q}$.
- Typically minimize $\mathrm{KL}\left(q \parallel p\right)$, which is equivalent to maximizing

$$\mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z})].$$

This objective has been the focus of most work in variational inference.

Operator Objectives

There are three ingredients:

1. An operator $O^{p,q}$ that depends on $p(\mathbf{z} \mid \mathbf{x})$ and $q(\mathbf{z})$.
2. A family of test functions $f \in \mathcal{F}$, where each $f(\mathbf{z}) : \mathbb{R}^d \to \mathbb{R}^d$.
3. A distance function $t(a) : \mathbb{R} \to [0, \infty)$.

These three ingredients combine to form an operator objective,

$$\sup_{f \in \mathcal{F}} t\left(\mathbb{E}_{q(\mathbf{z})}[(O^{p,q} f)(\mathbf{z})] \right).$$

It is the worst-case expected value among all functions $f \in \mathcal{F}$.

Operator Objectives

The goal is to minimize this objective,

$$\inf_{q \in \mathcal{Q}} \sup_{f \in \mathcal{F}} t(\ \mathbb{E}_{q(\mathbf{z})}[(O^{p,q} f)(\mathbf{z})]\ ).$$

In practice, we parameterize the variational family, $\{q(\mathbf{z}; \lambda)\}$. We also parameterize the test functions $\{f(\mathbf{z}; \theta)\}$ with a neural network.

$$\min_{\lambda} \max_{\theta} \ t(\ \mathbb{E}_{\lambda}[(O^{p,q} f_{\theta})(z)]\ ).$$

Operator Objectives

$$\sup_{f \in \mathcal{F}} t\big(\mathbb{E}_{q(\mathbf{z})}[(O^{p,q} f)(\mathbf{z})] \big).$$

To use these objectives for variational inference, we impose two conditions:

1. *Closeness*. Its minimum is achieved at the posterior,

$$\mathbb{E}_{p(\mathbf{z} \mid \mathbf{x})}[(O^{p,p} f)(\mathbf{z})] = 0 \text{ for all } f \in \mathcal{F}.$$

2. *Tractability*. The operator $O^{p,q}$—originally in terms of $p(\mathbf{z} \,|\, \mathbf{x})$ and $q(\mathbf{z})$—can be written in terms of $p(\mathbf{x}, \mathbf{z})$ and $q(\mathbf{z})$.

Operator Objectives: Examples

KL variational objective. The operator is

$$(O^{p,q} f)(z) = \log q(\mathbf{z}) - \log p(\mathbf{x}, \mathbf{z}) \quad \forall f \in \mathcal{F}.$$

With distance function $t(a) = a$, the objective is

$$\mathbb{E}_{q(\mathbf{z})}[\log q(\mathbf{z}) - \log p(\mathbf{x}, \mathbf{z})].$$

Operator Objectives: Examples

KL variational objective. The operator is

$$(O^{p,q} f)(\mathbf{z}) = \log q(\mathbf{z}) - \log p(\mathbf{x}, \mathbf{z}) \quad \forall f \in \mathcal{F}.$$

With distance function $t(a) = a$, the objective is

$$\mathbb{E}_{q(\mathbf{z})}[\log q(\mathbf{z}) - \log p(\mathbf{x}, \mathbf{z})].$$

Langevin-Stein variational objective. The operator is

$$(O^{p} f)(\mathbf{z}) = \nabla_z \log p(\mathbf{x}, \mathbf{z})^\top f(\mathbf{z}) + \nabla^\top f,$$

where $\nabla^\top f$ is the divergence of f. With distance function $t(a) = a^2$, the objective is

$$\sup_{f \in \mathcal{F}} \left(\mathbb{E}_{q(\mathbf{z})}[\nabla_z \log p(\mathbf{x}, \mathbf{z})^\top f(\mathbf{z}) + \nabla^\top f] \right)^2.$$

Operator Variational Inference

$$\min_{\lambda} \max_{\theta} t\left(\mathbb{E}_{\lambda}[(O^{p,q} f_{\theta})(z)] \right).$$

Fix $t(a) = a^2$; the case of $t(a) = a$ easily applies.

Operator Variational Inference

$$\min_{\lambda} \max_{\theta} t(\mathbb{E}_{\lambda}[(O^{p,q} f_{\theta})(z)]).$$

Fix $t(a) = a^2$; the case of $t(a) = a$ easily applies.

Gradient with respect to λ. (Variational approximation)

$$\nabla_{\lambda}\mathcal{L}_{\theta} = 2\, \mathbb{E}_{\lambda}[(O^{p,q} f_{\theta})(Z)]\, \nabla_{\lambda}\mathbb{E}_{\lambda}[(O^{p,q} f_{\theta})(Z)].$$

We use the score function gradient (Ranganath et al., 2014) and reparameterization gradient (Kingma & Welling, 2014).

Operator Variational Inference

$$\min_{\lambda} \max_{\theta} t\left(\mathbb{E}_{\lambda}[(O^{p,q} f_{\theta})(z)] \right).$$

Fix $t(a) = a^2$; the case of $t(a) = a$ easily applies.

Gradient with respect to λ. (Variational approximation)

$$\nabla_{\lambda}\mathcal{L}_{\theta} = 2\, \mathbb{E}_{\lambda}[(O^{p,q} f_{\theta})(Z)]\, \nabla_{\lambda}\mathbb{E}_{\lambda}[(O^{p,q} f_{\theta})(Z)].$$

We use the score function gradient (Ranganath et al., 2014) and reparameterization gradient (Kingma & Welling, 2014).

Gradient with respect to θ. (Test function)

$$\nabla_{\theta}\mathcal{L}_{\lambda} = 2\, \mathbb{E}_{\lambda}[(O^{p,q} f_{\theta})(z)]\, \mathbb{E}_{\lambda}[\nabla_{\theta} O^{p,q} f_{\theta}(z)].$$

We construct stochastic gradients with two sets of Monte Carlo estimates.

Characterizing Objectives: Data Subsampling

Stochastic optimization scales variational inference to massive data (Hoffman et al., 2013; Salimans & Knowles, 2013). The idea is to subsample data and scale the log-likelihood,

$$
\begin{aligned}
\log p(x_{1:n}, z_{1:n}, \beta) &= \log p(\beta) + \sum_{n=1}^{N} \Big[\log p(x_n \mid z_n, \beta) + \log p(z_n \mid \beta) \Big]. \\
&\approx \log p(\beta) + \frac{M}{N} \sum_{m=1}^{M} \Big[\log p(x_m \mid z_m, \beta) + \log p(z_m \mid \beta) \Big].
\end{aligned}
$$

Characterizing Objectives: Data Subsampling

Stochastic optimization scales variational inference to massive data Hoffman et al., 2013; Salimans & Knowles, 2013). The idea is to subsample data and scale the log-likelihood,

$$
\begin{aligned}
\log p(x_{1:n}, z_{1:n}, \beta) &= \log p(\beta) + \sum_{n=1}^{N} \Big[\log p(x_n \mid z_n, \beta) + \log p(z_n \mid \beta) \Big]. \\
&\approx \log p(\beta) + \frac{M}{N} \sum_{m=1}^{M} \Big[\log p(x_m \mid z_m, \beta) + \log p(z_m \mid \beta) \Big].
\end{aligned}
$$

One class of operators which admit data subsampling are linear operators with respect to $\log p(\mathbf{x}, \mathbf{z})$.

The LS and KL operators are examples in this class. (An operator for f-divergences is not.)

Characterizing Objectives: Variational Programs

Recent advances in variational inference aim to develop expressive approximations, such as with transformations (Rezende & Mohamed, 2015; Tran et al., 2015; Kingma et al., 2016) and auxiliary variables (Salimans et al., 2015; Tran et al., 2016; Ranganath et al., 2016).

In variational inference, our design of the variational family $q \in \mathcal{Q}$ is limited by a tractable density.

Characterizing Objectives: Variational Programs

We can design operators that do not depend on q, $O^{p,q} = O^p$, such as the LS objective

$$\sup_{f \in \mathcal{F}} \left(\, \mathbb{E}_{q(\mathbf{z})}[\nabla_z \log p(\mathbf{x}, \mathbf{z})^\top f(\mathbf{z}) + \nabla^\top f] \, \right)^2 .$$

The class of approximating families is much larger, which we call *variational programs*.

Consider a generative program of latent variables,

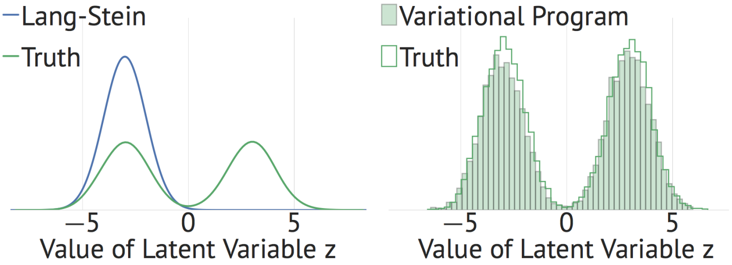
$$\epsilon \sim \mathrm{Normal}(0, 1), \quad \mathbf{z} = G(\epsilon; \lambda),$$

where G is a neural network. The program is differentiable and generates samples for $\mathbf{z}$. Moreover, its density does not have to be tractable.

Experiments

Variational program:

1. Draw $\epsilon, \epsilon' \sim \mathrm{Normal}(0, 1)$.
2. If $\epsilon' > 0$, return $G_1(\epsilon)$; else if $\epsilon' \leq 0$, return $G_2(\epsilon)$.



1-D Mixture of Gaussians. LS with a Gaussian family fits a mode. LS with a variational program approaches the truth.

Experiments

We model binarized MNIST, $\mathbf{x}_n \in \{0, 1\}^{28\times 28}$, with

$$\mathbf{z}_n \sim \mathrm{Normal}(0, 1),$$
$$\mathbf{x}_n \sim \mathrm{Bernoulli}(\mathrm{logistic}(\mathbf{z}_n^\top \mathbf{W} + \mathbf{b})),$$

where $\mathbf{z}_n$ has latent dimension 10 and with parameters $\{\mathbf{W}, \mathbf{b}\}$.

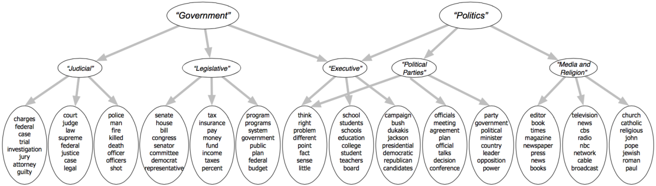
At test time, we throw away half the pixels and impute them using different objectives. We compare the log-likelihood of the completed image.

Inference method	Completed data log-likelihood
Mean-field Gaussian + KL$(q\|\|p)$	-59.3
Mean-field Gaussian + LS	-75.3
Variational Program + LS	-58.9

Table: The variational program performs better than KL without directly optimizing for likelihoods.

How do we design efficient and flexible software for generative modeling?

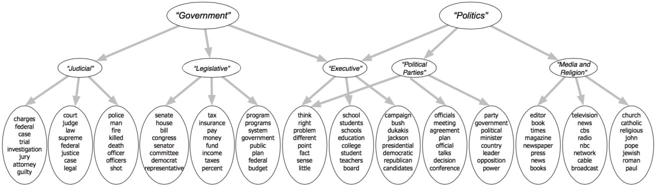
Motivation



Hierarchy of topics found in 166K articles from the New York Times

What existing probabilistic programming languages enable this analysis?

Motivation



Hierarchy of topics found in 166K 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	BBVI (w/ mini-batches + inference networks)
Infer.NET	VMP

Punchline: We need the graph structure.

Edward is a library for probabilistic modeling, inference, and criticism.

- It extends the formalism of computational graphs to generative models and their inference.
- Only two abstractions are built on top of TensorFlow: random variables and inference classes.

Model: Random Variables

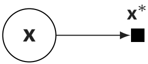
A random variable **x** is an *object* parameterized by tensors θ^*.

```
# 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)
```

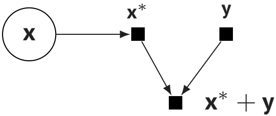
It is equipped with methods such as `log_prob()` and `sample()`.

Model: Random Variables

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

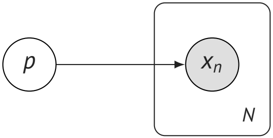


This enables ops on the computational graph. They operate on $\mathbf{x}^*$.




```
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



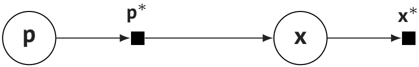
$$p \sim \text{Beta}(1, 1)$$
$$x_n \sim \text{Bernoulli}(p)$$

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

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

Model: Directed Graphical Models

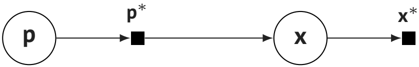
The model defines a computational graph.




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

Model: Directed Graphical Models

The model defines a computational graph.



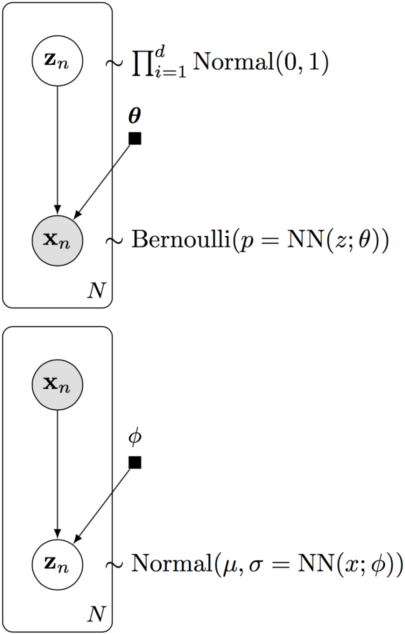

```
p = Beta(a=1.0, b=1.0)
x = Bernoulli(p=tf.ones(N) * p)
```

Running the graph for **x** will:

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

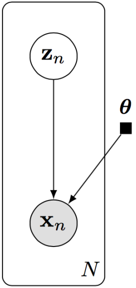
Directed structure is exposed in the computational graph. We can now write model-specific algorithms (and generic algorithms).

Example: Variational Auto-encoder for Binarized MNIST

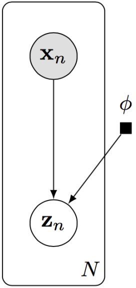


(Kingma & Welling, 2014; Rezende et al., 2014)

Example: Variational Auto-encoder for Binarized MNIST



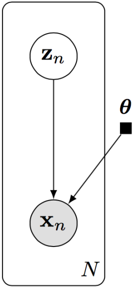

```
z = Normal(mu=tf.zeros([N, d]), sigma=tf.ones([N, d]))
hidden = Dense(64, activation='relu')(z.value())
x = Bernoulli(logits=Dense(28 * 28)(hidden))
```

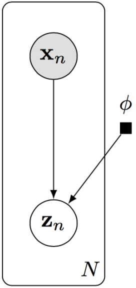



```
x_ph = ed.placeholder(tf.float32, [N, 28 * 28])
hidden = Dense(64, activation='relu')(x_ph)
qz = Normal(mu=Dense(d)(hidden),
            sigma=Dense(d, activation='softplus')(hidden))
```

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

Example: Variational Auto-encoder for Binarized MNIST

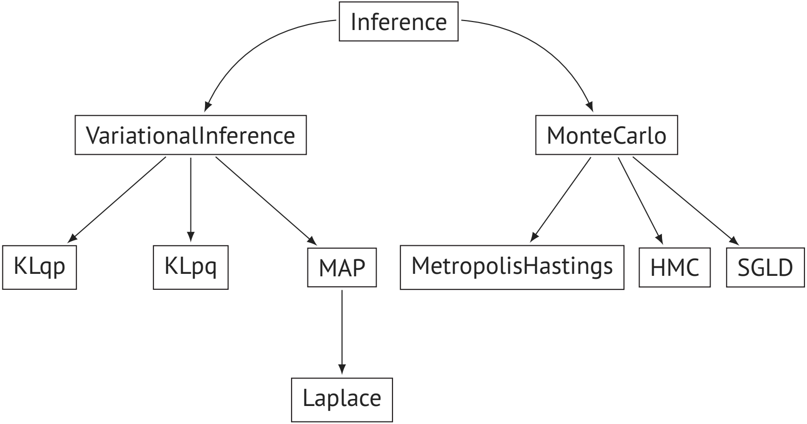





```
N = 1000
d = 10

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

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

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

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

Inference



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

Inference

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

```
qbeta = Normal(mu=tf.Variable(tf.zeros([d])),
               sigma=tf.nn.softplus(tf.Variable(tf.ones[d])))
qz = Normal(mu=tf.Variable(tf.zeros([d])),
            sigma=tf.nn.softplus(tf.Variable(tf.ones[d])))

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

```
T = int(1e4)  # number of samples
qbeta = Empirical(params=tf.Variable(tf.zeros([T, d])))
qz = Empirical(params=tf.Variable(tf.zeros([T, d])))

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

Inference: Composing Inference

EM algorithm.

```
qbeta = PointMass()
qz = RandomVariable()
inference_e = ed.KLqp({z: qz}, data={x: x_train, beta: qbeta})
inference_m = ed.MAP({beta: qbeta}, data={x: x_train, z: qz})

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

Summary

1. **Tradeoffs.** Developed a language to tradeoff statistical and computational properties during inference.
2. **Software.** Developed a generative modeling language on computational graphs, with model structure exposed to the user.

   Developed a language around inference, including both model-specific and generic algorithms.

Key References

- R. Ranganath, J. Altosaar, **D. Tran**, and D.M. Blei. Operator variational inference. In *Neural Information Processing Systems*, 2016.
  (arXiv this week)
- **D. Tran**, A. Kucukelbir, A. Dieng, M. Rudolph, D. Liang, and D.M. Blei. Edward: A library for probabilistic modeling, inference, and criticism.
  `edwardlib.org`
  (arXiv this week)
- **D. Tran**, M. Hoffman, K. Murphy, E. Brevdo, R. Saurous, and D.M. Blei. Generative modeling and inference on computational graphs.
  (ICLR submission)