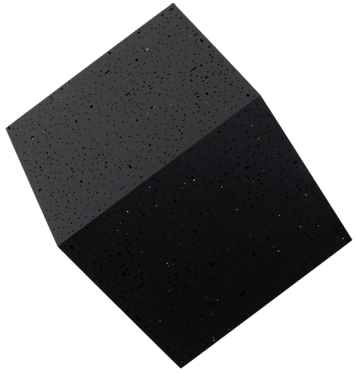
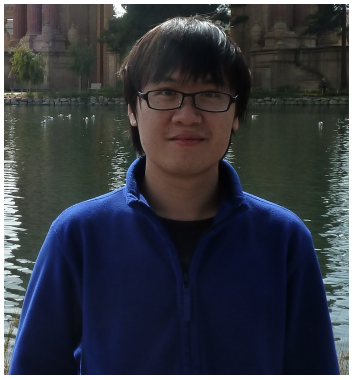
Probabilistic Programming with GP Applications

Dustin Tran
Columbia University, OpenAI

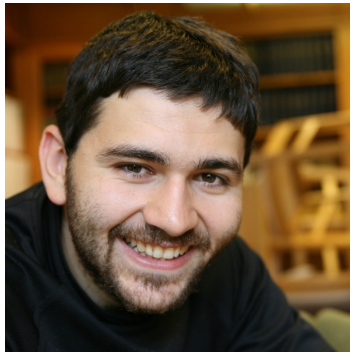


Alp Kucukelbir

Adji Dieng



Dawen Liang



Eugene Brevdo

Maja Rudolph

Matt Hoffman

Rajesh Ranganath

Andrew Gelman

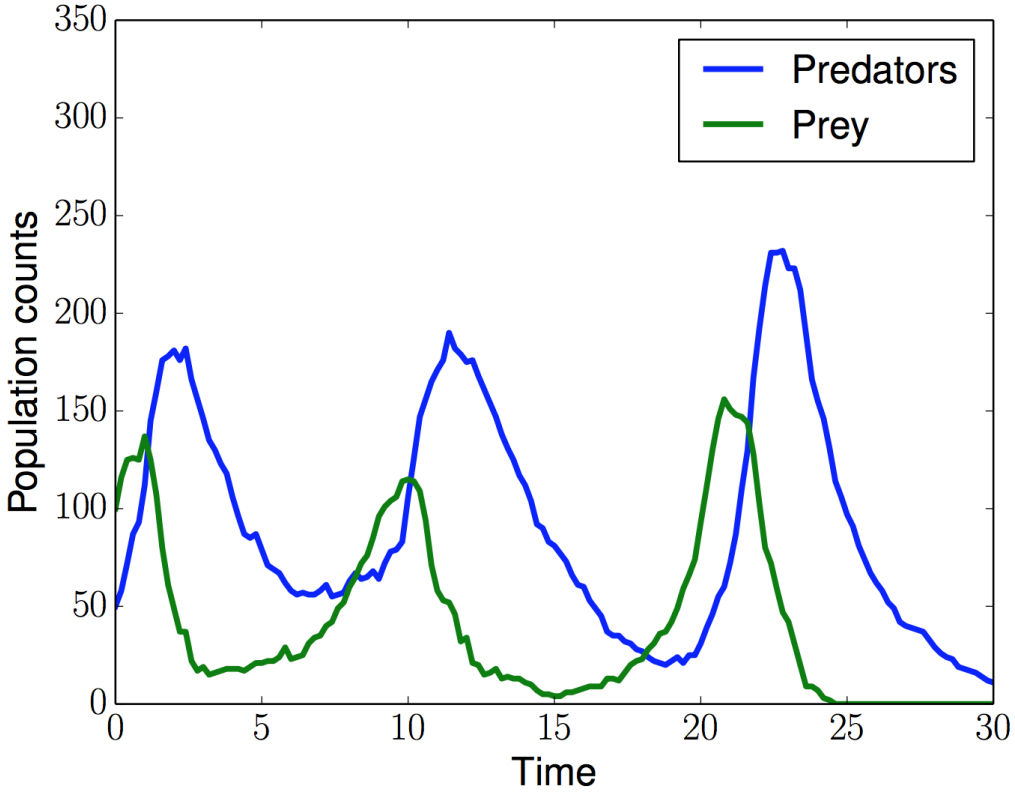
David Blei

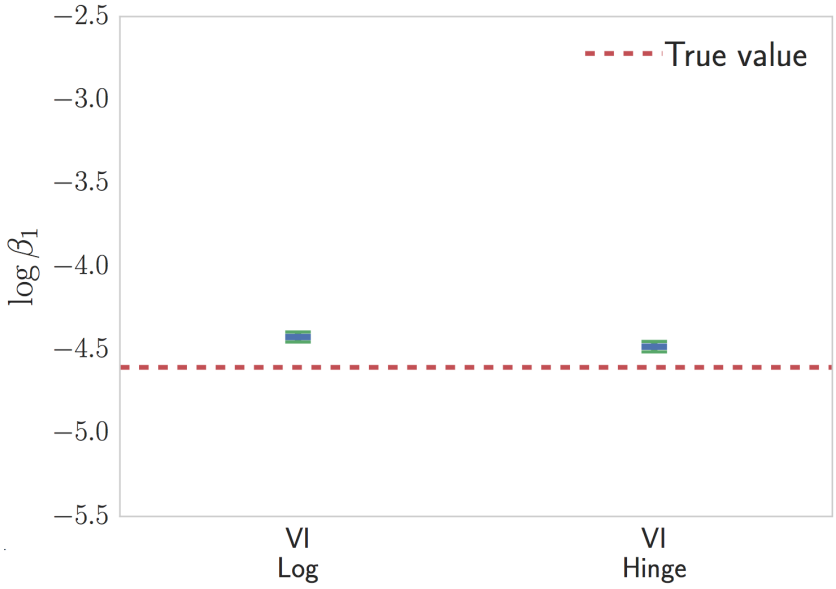
Kevin Murphy

Rif Saurous

Exploratory analysis of 1.7M taxi trajectories, in Stan

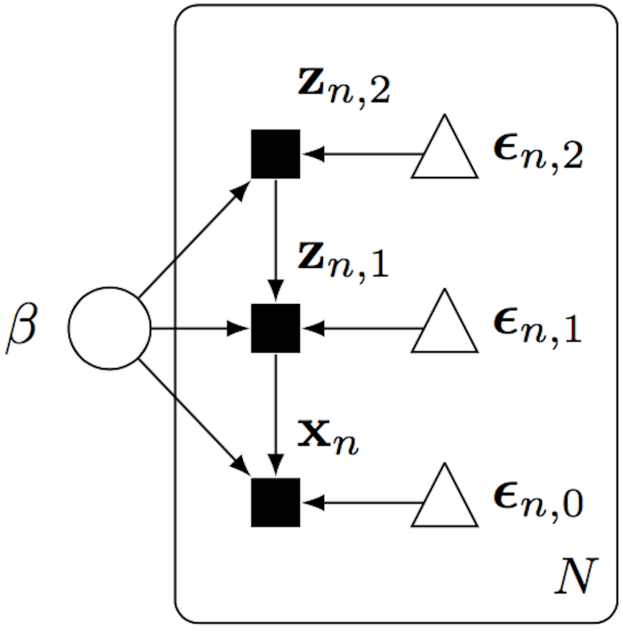
[Kucukelbir+ 2017]

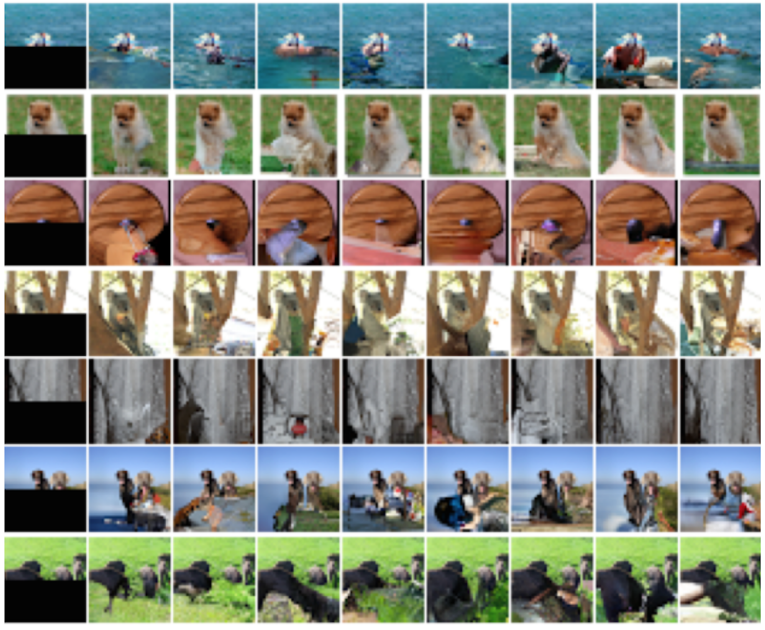




Simulators of 100K time series in ecology, in Edward

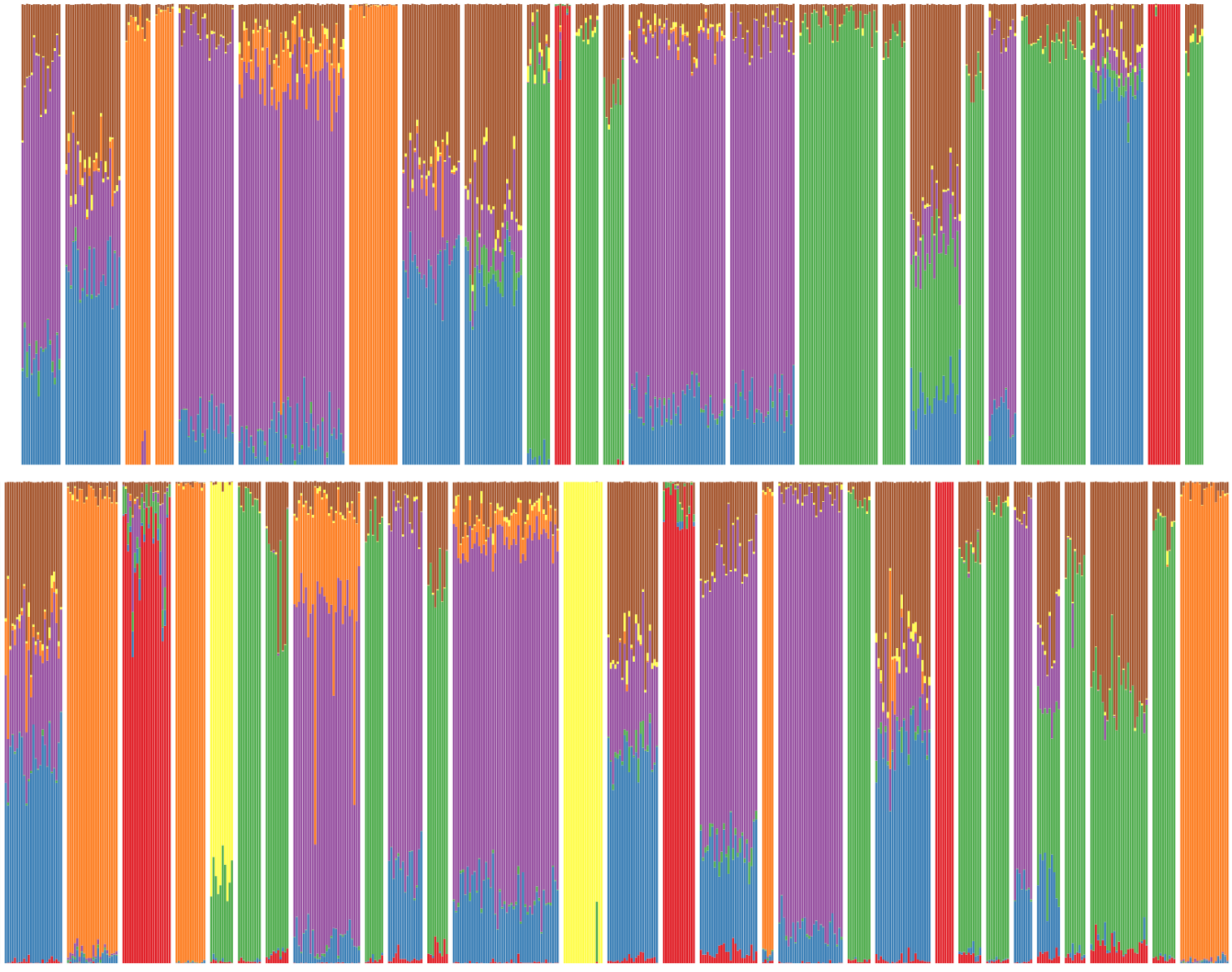
[Tran+ 2017]





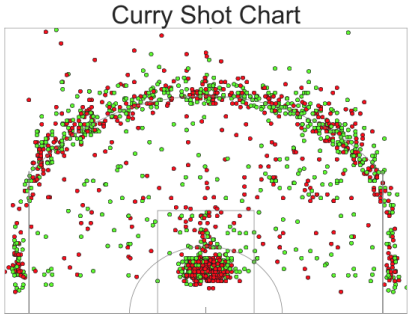
Generation & compression of 10M colored 32x32 images, in Edward

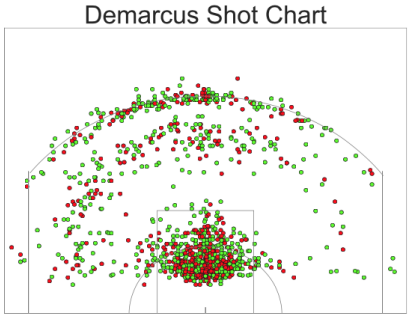
[Tran+ 2017; fig from Van der Oord+ 2016]

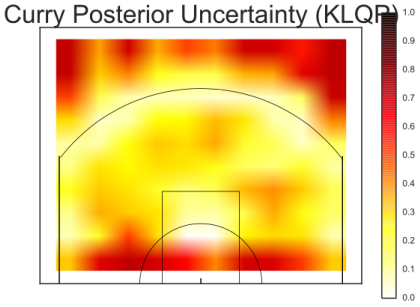


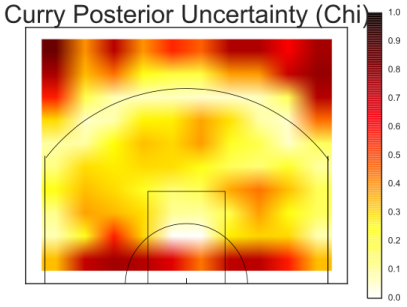
Cause and effect of 1.6B genetic measurements, in Edward

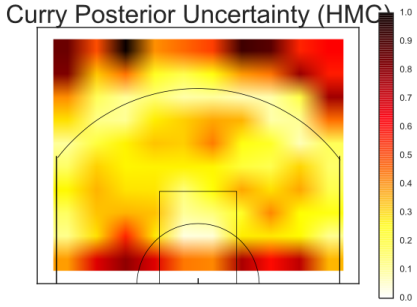
[in preparation; fig from Gopalan+ 2017]

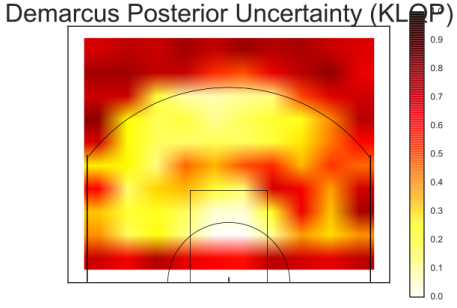


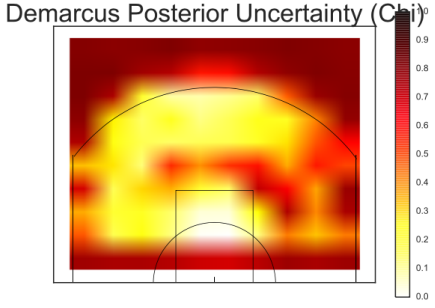


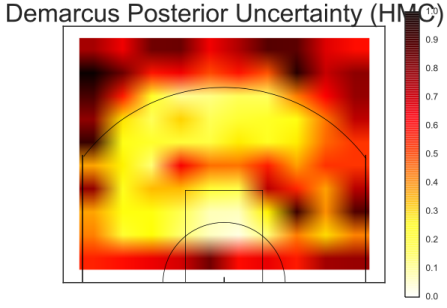












Spatial analysis of 150,000 shots from 308 NBA players, in Edward

[Dieng+ 2017]

What is probabilistic programming?

Probabilistic programs reify models from mathematics to physical objects.

- Each model is equipped with memory (“bits”, floating point, storage) and computation (“flops”, scalability, communication).

Anything you do lives in the world of probabilistic programming.

- Any computable model.

  ex. graphical models; neural networks; SVMs; stochastic processes.

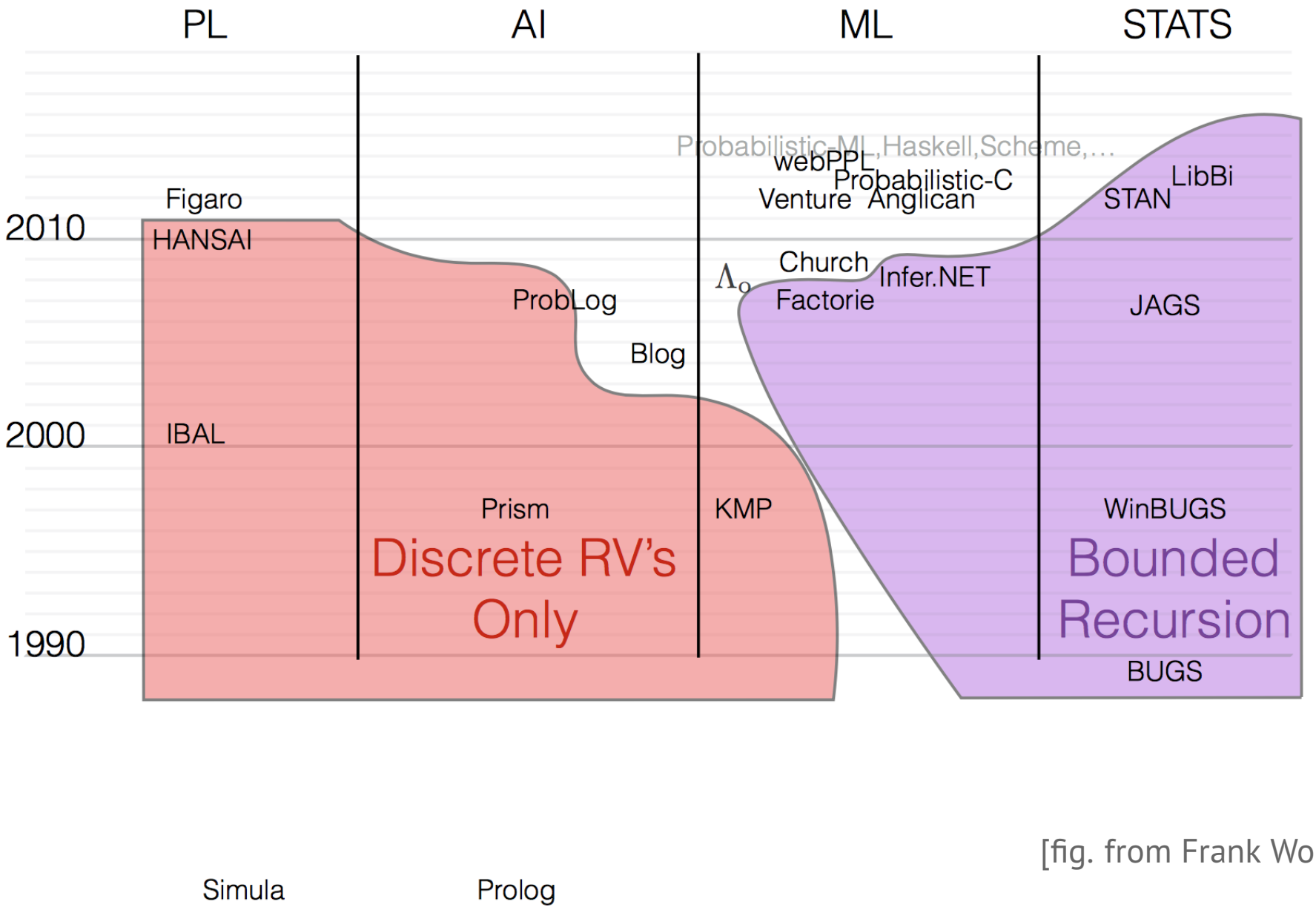
- Any computable inference algorithm.

  ex. automated inference; model-specific algorithms; inference within inference (learning to learn).

- Any computable application.

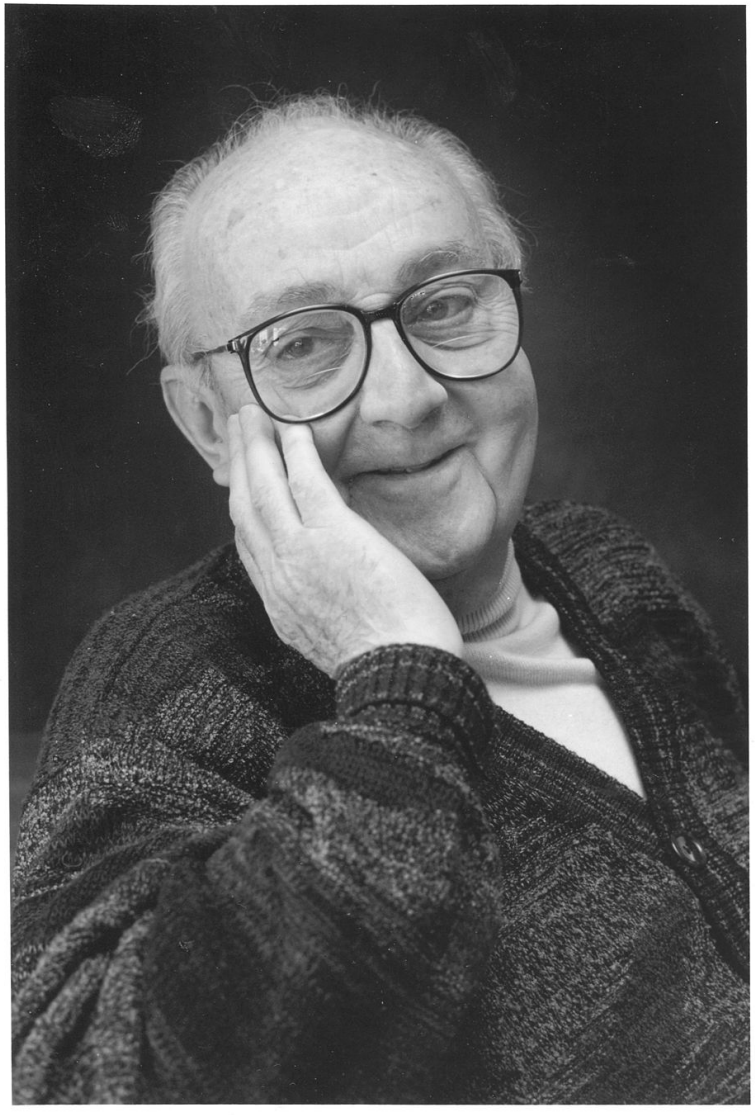
  ex. exploratory analysis; object recognition; code generation; causality.

Languages and Systems



[fig. from Frank Wood]

George E.P. Box (1919 - 2013)

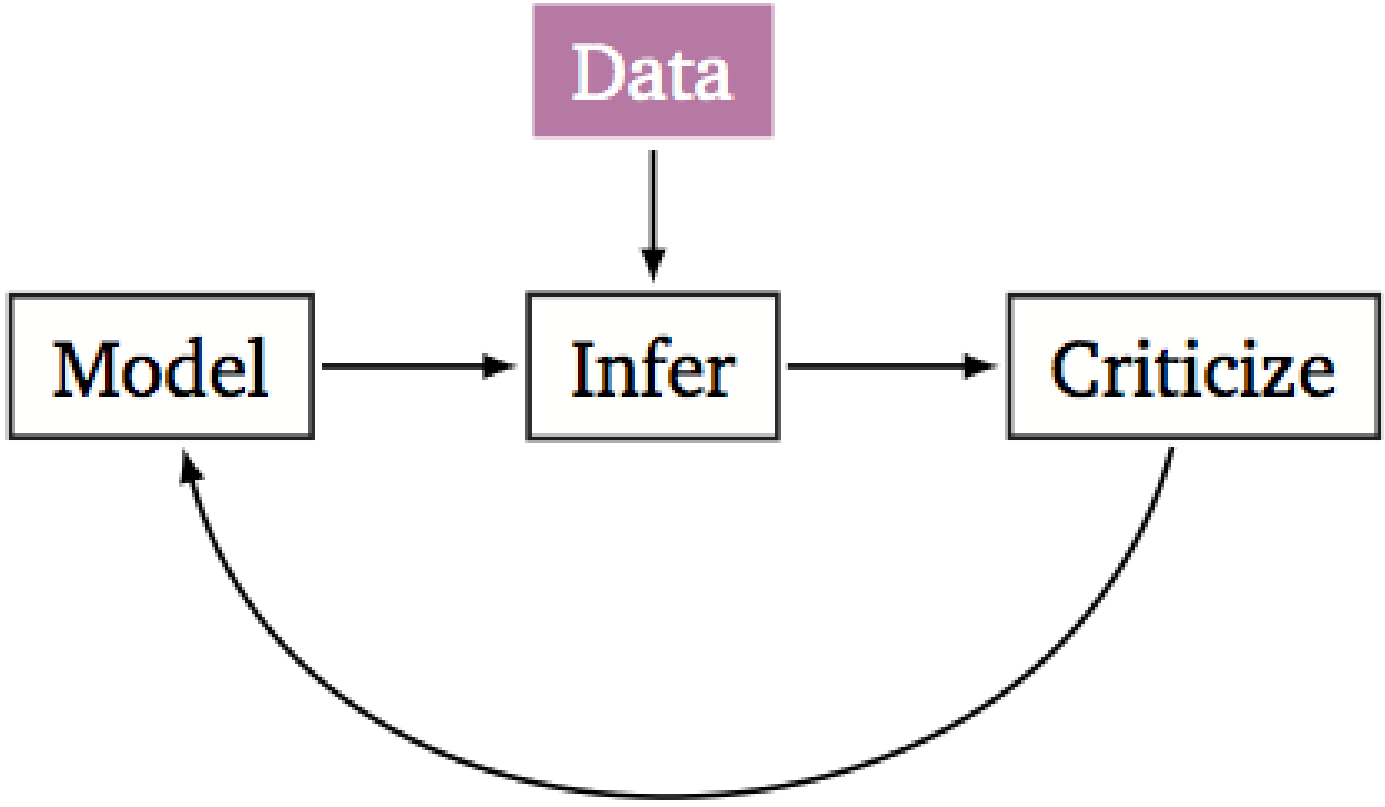


An iterative process for science:

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

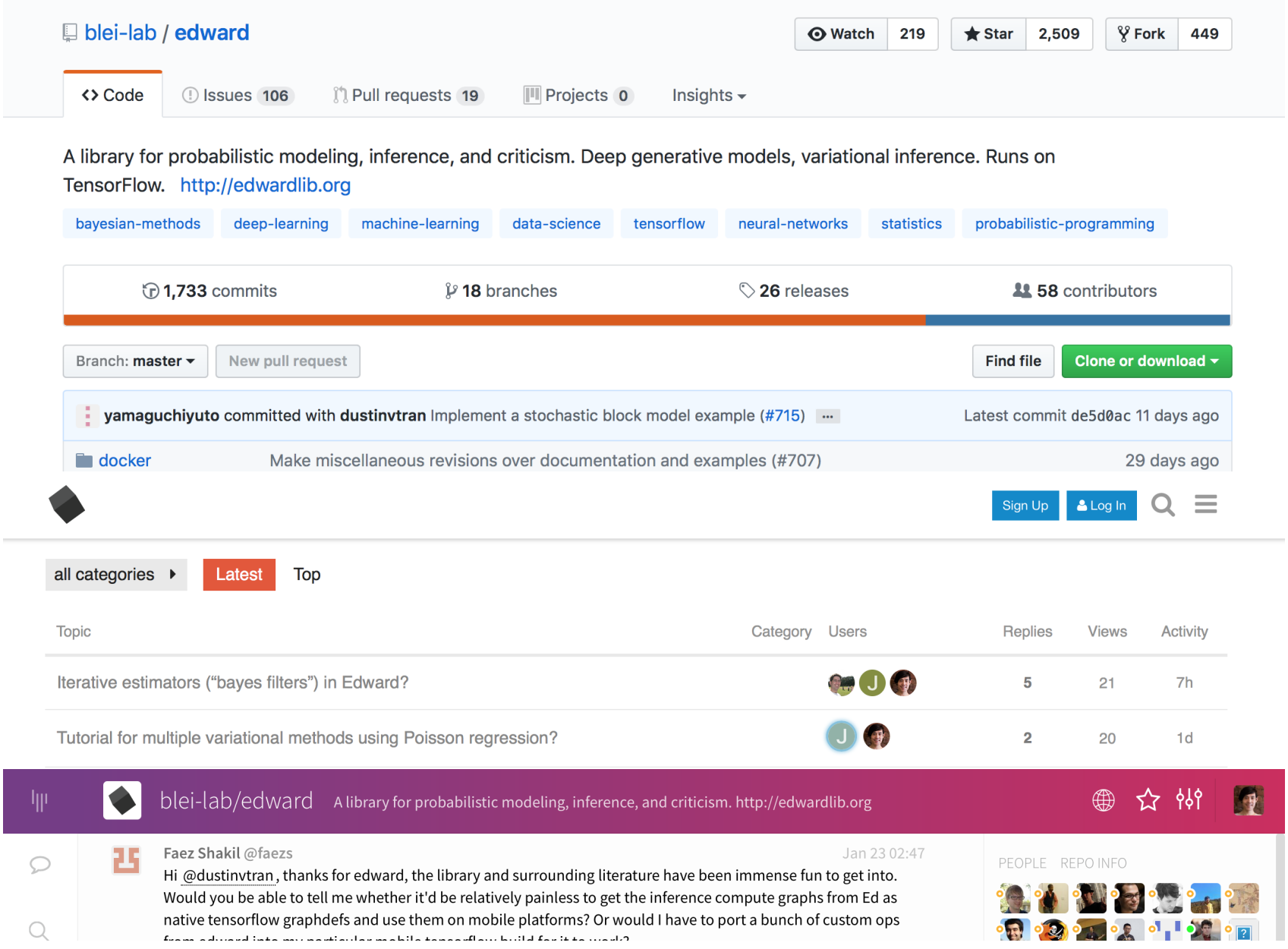
[Box & Hunter 1962, 1965; Box & Hill 1967; Box 1976, 1980]

Box's Loop



Edward is a library designed around this loop.

[Box 1976, 1980; Blei 2014]



We have an active community of several thousand users & many contributors.

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

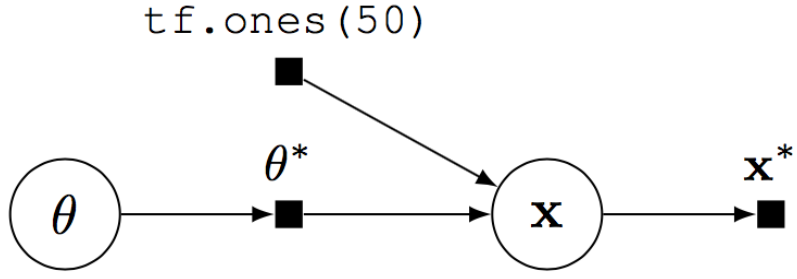
Example: Beta-Bernoulli

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)
```



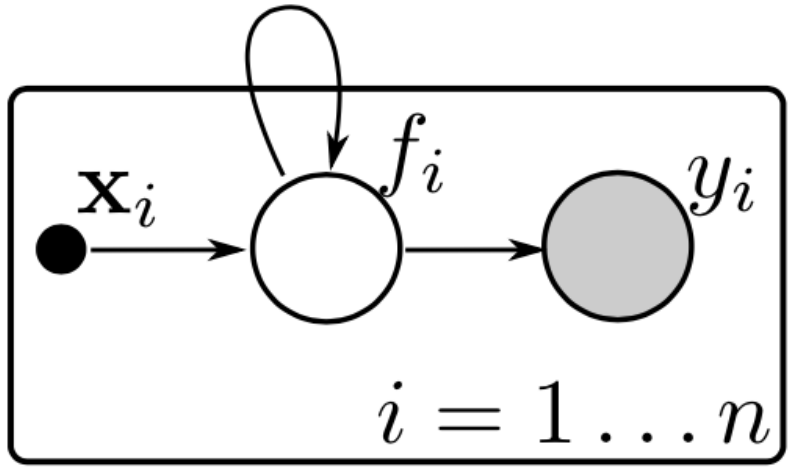
Fetching **x** from the graph generates a binary vector of 50 elements.

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

How does probabilistic programming help GPs for modeling?

- GPs are a module to compare against other model classes in experiments.
- GPs can be combined with other model classes and advances (e.g., batch norm, dropout, convolutions, recurrence).
- Deep GPs, deep kernels, and hierarchical GPs.
- Flexible priors over kernel hyperparameters and non-Gaussian likelihoods.

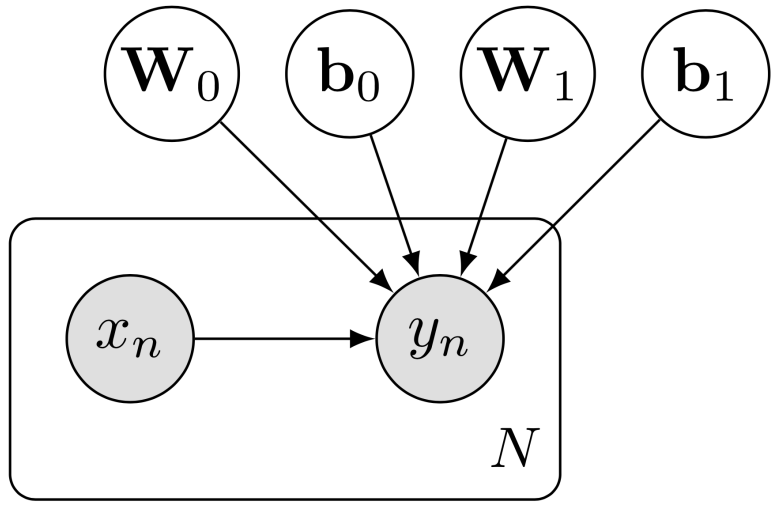
Example: Gaussian process classification




```
X = tf.placeholder(tf.float32, [N, D])
f = MultivariateNormalTriL(loc=tf.zeros(N),
                           scale_tril=tf.cholesky(rbf(X)))
y = Bernoulli(logits=f)
```

[Rasmussen & Williams, 2006; fig from Hensman+ 2013]

Example: Bayesian neural network for classification

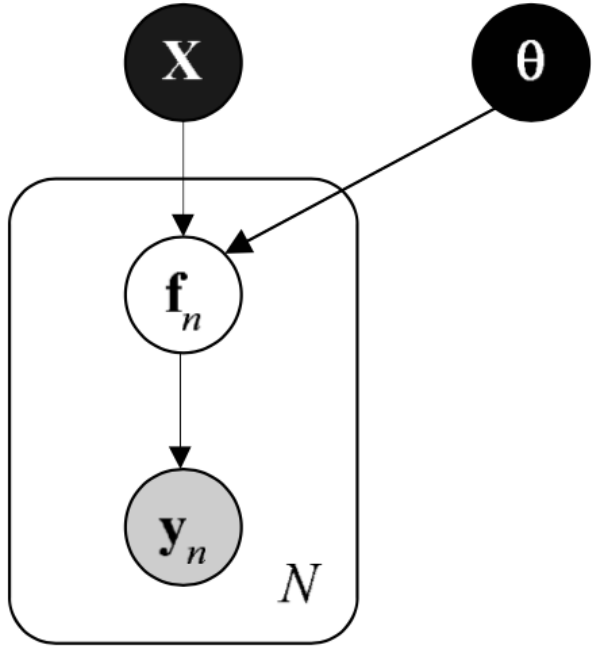



```
W_0 = Normal(mu=tf.zeros([D, H]), sigma=tf.ones([D, H]))
W_1 = Normal(mu=tf.zeros([H, 1]), sigma=tf.ones([H, 1]))
b_0 = Normal(mu=tf.zeros(H), sigma=tf.ones(L))
b_1 = Normal(mu=tf.zeros(1), sigma=tf.ones(1))

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)
```

[Denker+ 1987; MacKay 1992; Hinton & Van Camp, 1993; Neal 1995]

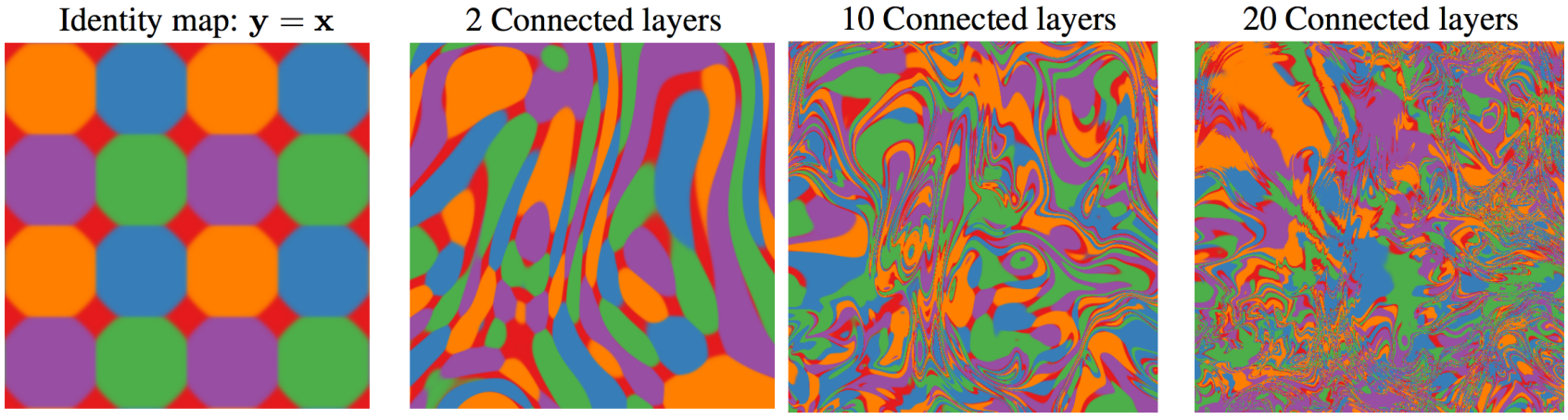
Example: Gaussian process latent variable model




```
X = Normal(loc=tf.zeros([N, Q]), scale=tf.ones([N, Q]))
f = MultivariateNormalTriL(loc=tf.zeros([N, D]),
                           scale_tril=tf.cholesky(rbf(X)))
y = Bernoulli(logits=f)
```

[Lawrence 2005; Titsias & Lawrence 2010]

Example: Deep Gaussian process




```
X = Normal(loc=tf.zeros([N, Q]), scale=tf.ones([N, Q]))
h1 = MultivariateNormalTriL(loc=tf.zeros([N, H1]),
                            scale_tril=tf.cholesky(rbf(X)))
h2 = MultivariateNormalTriL(loc=tf.zeros([N, H2]),
                            scale_tril=tf.cholesky(rbf(h1)))
f = MultivariateNormalTriL(loc=tf.zeros([N, D]),
                           scale_tril=tf.cholesky(rbf(h2)))
y = Bernoulli(logits=f)
```

[Damianou & Lawrence 2010; fig from Duvenaud+ 2014]

Example: Cox process

[Demo]

github.com/blei-lab/edward/blob/master/examples/cox_process.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.

How does probabilistic programming help GPs for inference?

- GPs with flexible inference strategies: VI with inducing variables; SVI for GPs; EP; MML; CCD.
- GPs on multi-machine, multi-device environments; training/test with float64, float32, int8; autodiff.
- GPs as posterior approximations.
- GPs for inference within inference (Bayes Opt).

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.

Inference: Composing Inference

Core to Edward's design is that inference can be written as a collection of separate inference programs.

For example, here is variational EM.

```
qbeta = PointMass(params=tf.Variable(tf.zeros([K, D])))
qz = Categorical(logits=tf.Variable(tf.zeros[N, K]))

inference_e = ed.VariationalInference({z: qz}, data={x: x_data, beta: qbeta})
inference_m = ed.MAP({beta: qbeta}, data={x: x_data, z: qz})

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

We can also write message passing algorithms, which work over a collection of local inference problems. This includes expectation propagation.

[Neal & Hinton 1993; Minka 2001; Gelman+ 2017]

Summary

1. Edward is a probabilistic programming system 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 on large data or multi-device / multi-machine environments.

Current directions

We are extending Edward with new semantics.

1. Benchmarks across modeling & inference applications.
2. Dynamic computational graphs.
3. A one-line API for loading standard data sets in machine learning.

We are applying Edward for AI and scientific research.

1. Causal models for genome wide association studies. (Blei)
2. Alignment & data efficiency in language. (OpenAI)
3. AGI (Solomonoff induction) by learning probabilistic programs. (OpenAI)

References

edwardlib.org

Two NIPS workshops this year:

- Approximate Bayesian Inference (approximateinference.org).
- Bayesian Deep Learning (bayesiandeeplearning.org).

Two NIPS papers this year:

- **D. Tran**, R. Ranganath, and D.M. Blei.
  Deep and hierarchical implicit models.
- A. Dieng, **D. Tran**, R. Ranganath, and D.M. Blei.
  The χ-divergence for approximate inference.