Deep Probabilistic Programming

Unicode Research

Deep Gandhi
Jash Mehta
Jay Gala

Standard Deep Learning

[Rowan's slide]

- Computer vision, natural language processing, speech recognition, etc
- Feedforward, Recurrent, Convolutional . . .
- SGD, Backprop, dropout

[Rowan’s slide]

Issues with Standard Deep Learning

- Computes point estimates
- Overly confident
- Overfitting
- Hyperparameter Tuning

[Tran's slide]

Probabilistic Machine Learning

A probabilistic model is a joint distribution of *hidden variables z* and *observed variables x*

$$p(z, x)$$

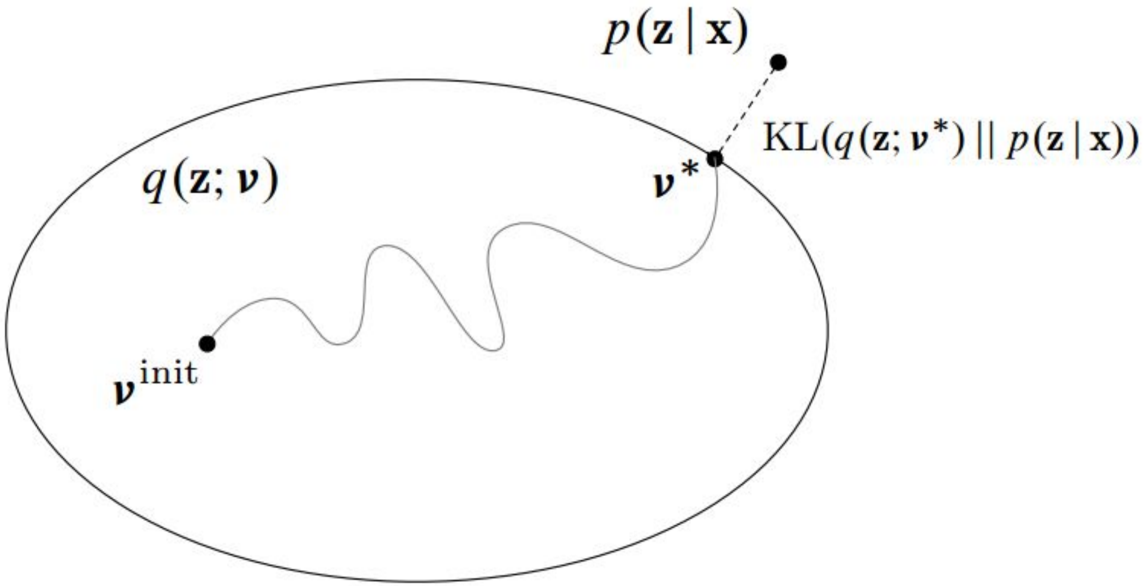
Inference about the unknowns is through the *posterior*, the conditional distribution of the hidden variables given the observations

$$p(z|x) = \frac{p(z,x)}{p(x)}$$

Inference is intractable due to the denominator!

Variational Inference

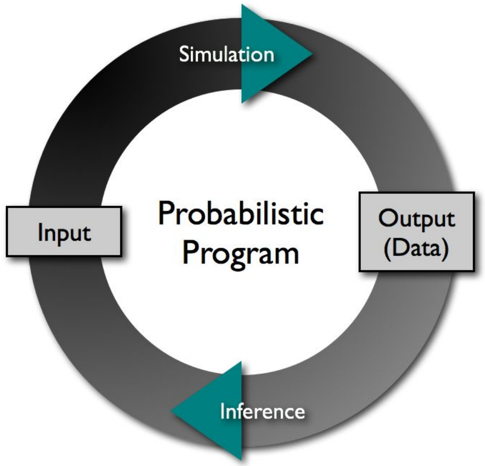
[Tran's slide]


$p(\mathbf{z} \mid \mathbf{x})$
$\mathrm{KL}(q(\mathbf{z}; \boldsymbol{\nu}^*) \,||\, p(\mathbf{z} \mid \mathbf{x}))$
$q(\mathbf{z}; \boldsymbol{\nu})$
$\boldsymbol{\nu}^*$
$\boldsymbol{\nu}^{\text{init}}$

Approximate posterior inference

Probabilistic Programming

*Probabilistic models as **programs** to generate **samples***



[Image credits: O'Reilly]

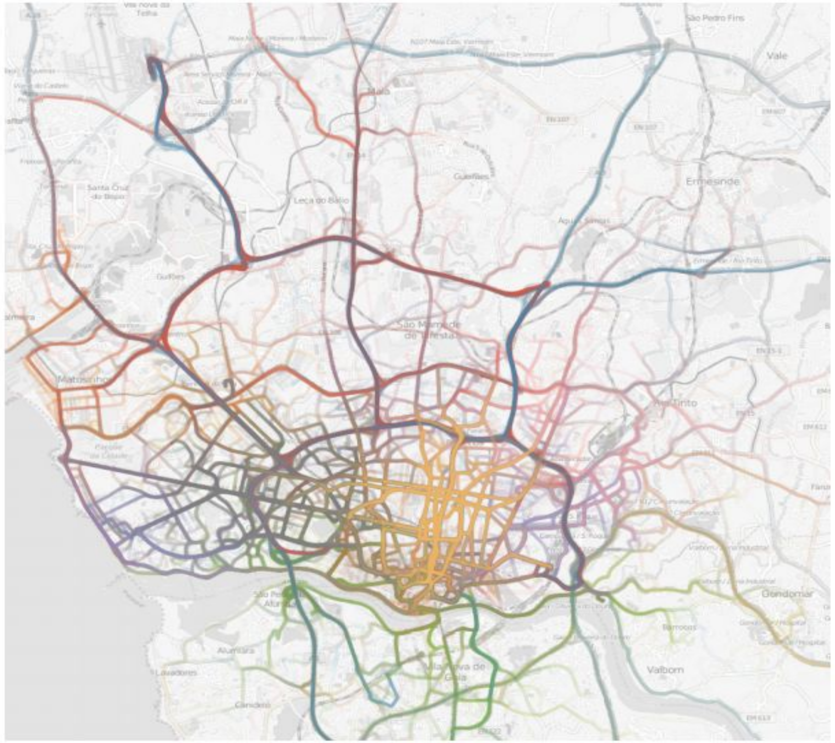
PPLs: Church, Venture, Anglican, Stan, PyMC3, Edward . . .

Some applications ...

1. Exploratory analysis of 1.7M taxi trajectories, in Stan

1. Cause and effect of 1.6B genetic measurements, in Edward

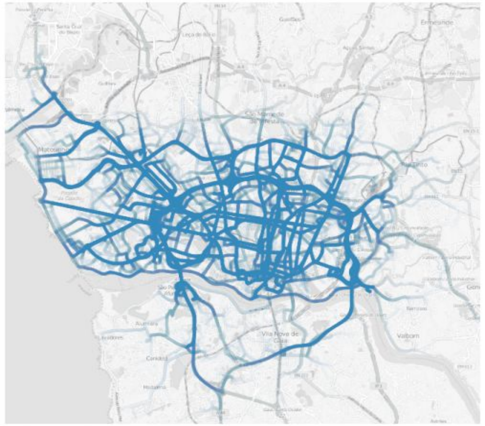
1. Spatial analysis of 150,000 shots from 308 NBA players, in Edward



A visualization of fifty thousand randomly sampled taxi trajectories. The colors represent thirty Gaussian mixtures and the trajectories associated with each.

[Tran's slide]

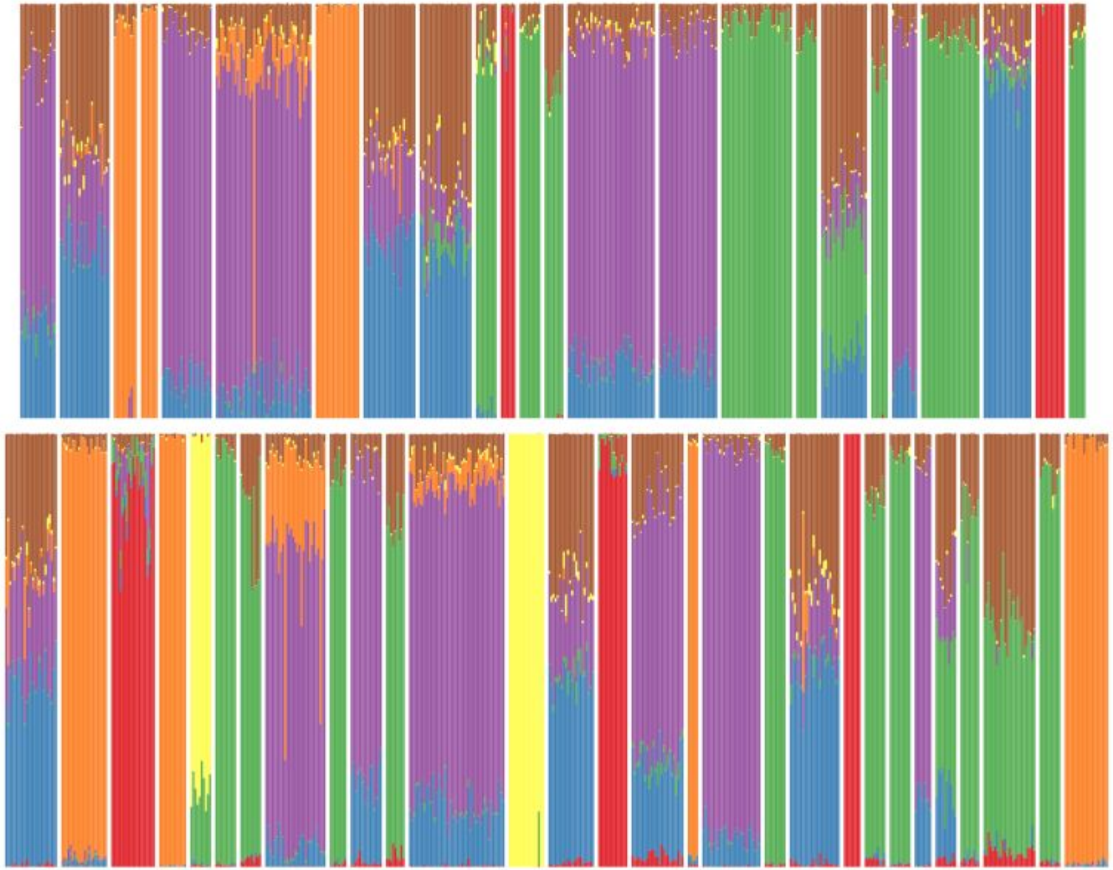
(a) Trajectories that take the inner bridges.



(b) Trajectories that take the outer bridges.

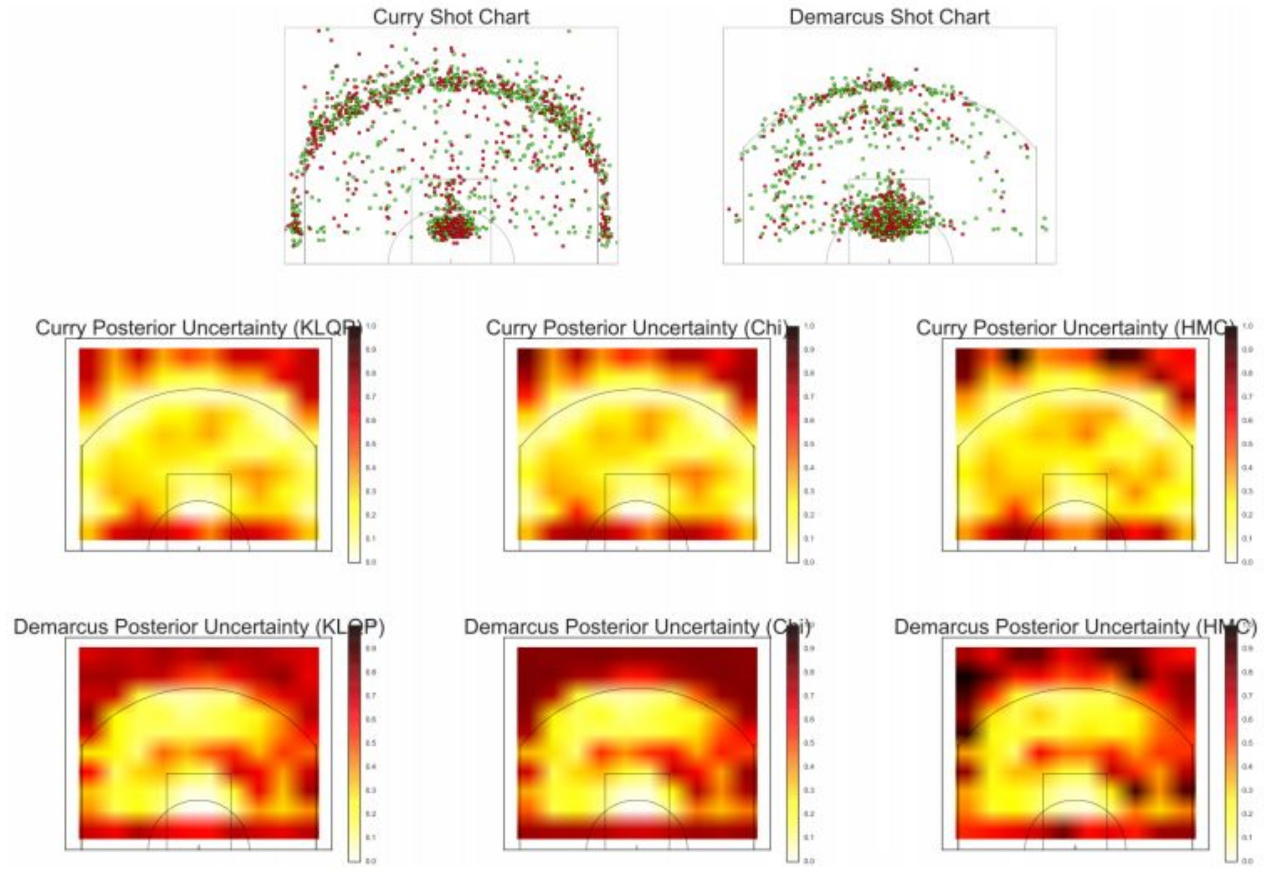
[Kucukelbir+ 2017]

[Tran's slide]



This figure shows the cause and effect of 1.6B genetic measurements, **in Edward**

[Gopalan+ 2017]



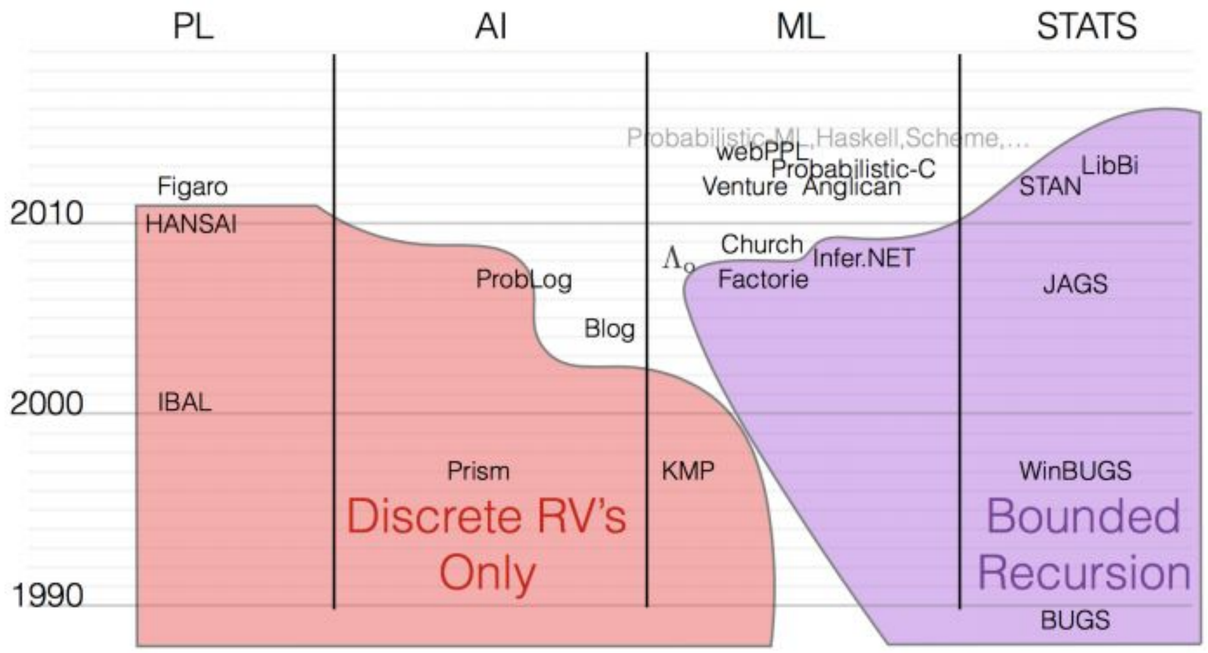
[Tran's slide]

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

[Dieng+ 2017]

[Tran’s slide]

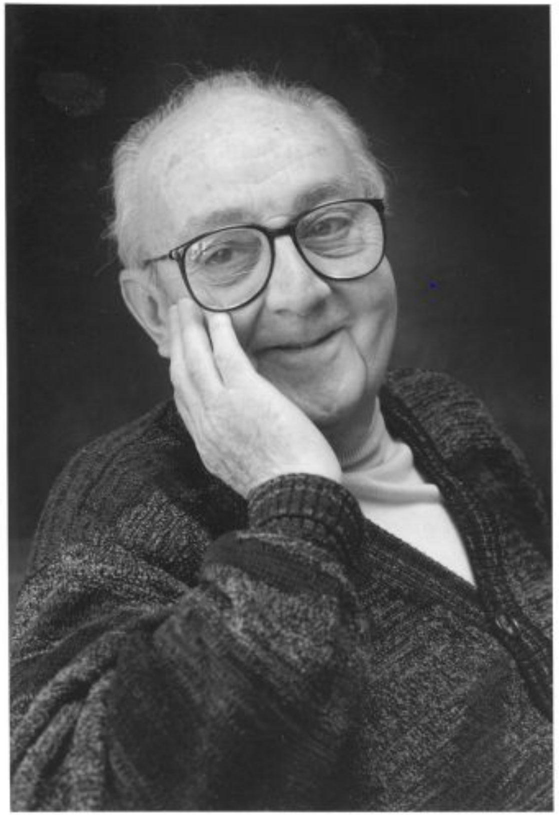
Languages and Systems



[Frank Wood]

George E.P. Box (1919 - 2013)

[Tran's slide]



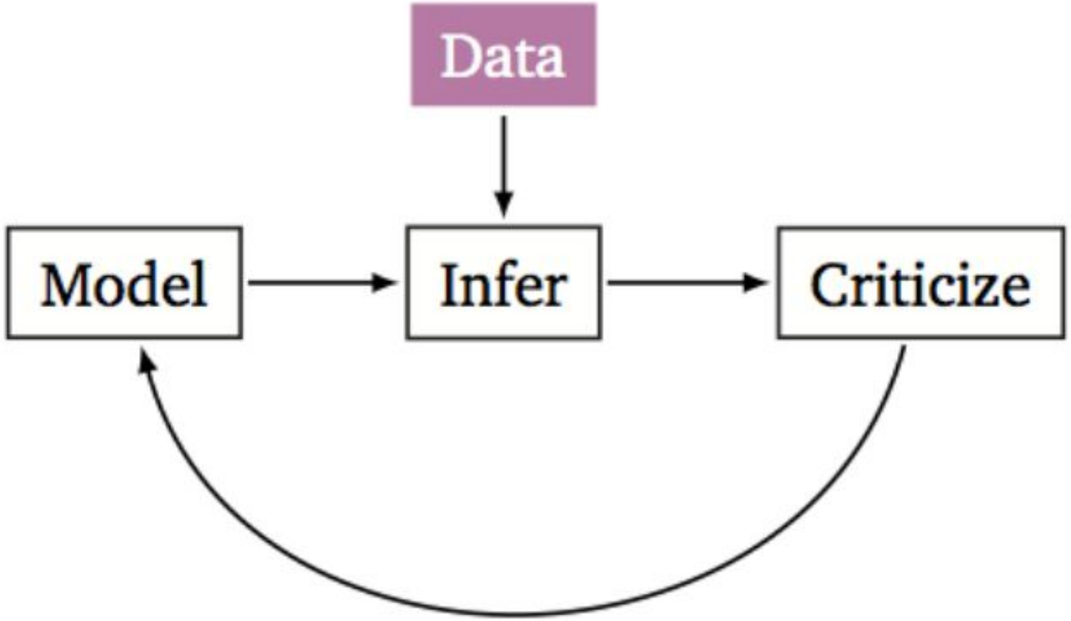
An iterative process for science:

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]

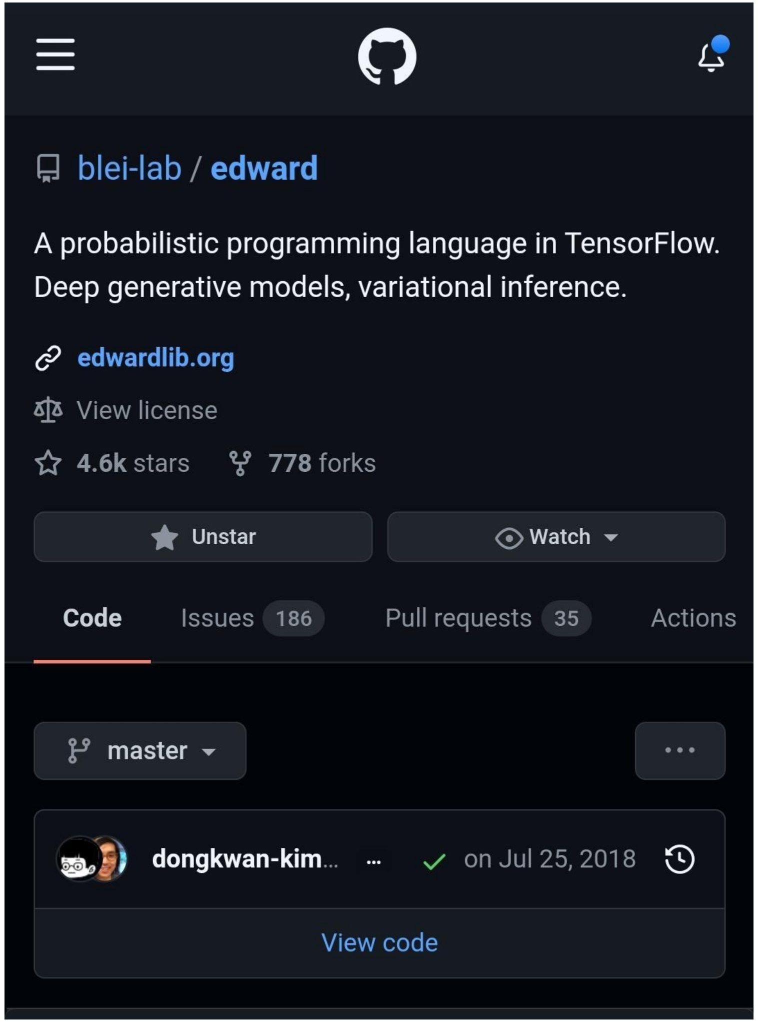
Box's Loop

[Tran's slide]



Edward is a library designed around this loop.

[Box 1976, 1980; Blei 2014]


blei-lab / edward

A probabilistic programming language in TensorFlow. Deep generative models, variational inference.

edwardlib.org

View license

4.6k stars 778 forks

- Python library.
- probabilistic modeling, inference, and criticism.
- a testbed for fast experimentation and research with probabilistic models
- fuses three fields:
  - Bayesian statistics and ML
  - deep learning, and
  - probabilistic programming.

[Github Link]

About Edward

- Turing-complete.

- As flexible and computationally efficient as traditional deep learning

- Edward defines 2 compositional representations:
    - Random variables
    - Inference

- Treats inference as first class citizen on par with modelling

- Incurs no runtime overhead

[Tran et al, 2017]

Related Work

Expressiveness of the language	Computational Efficiency of inference
- Generic inference engine but scales poorly wrt model and data size - Church	- Restricted to specific class of models and inference optimized for this class - Infer.NET: fast message passing for graphical models - Augur: data parallelism with GPUs for gibbs sampling
Edward bridges the gap !!!	

[Tran et al, 2017]

Related Work

Venture and Anglican:

- **Edward builds on designing inference as collection of local inference problems**
- Edward supports programmable posterior approximations, inference models, data subsampling which they don't

WebPPL:

- Features amortized inference like Edward
- Does not reuse model's representations
- **Edward can reuse the modeling representation as part of inference**

Edward builts on the idea to compose not only inference within modelling but also modeling within inference.

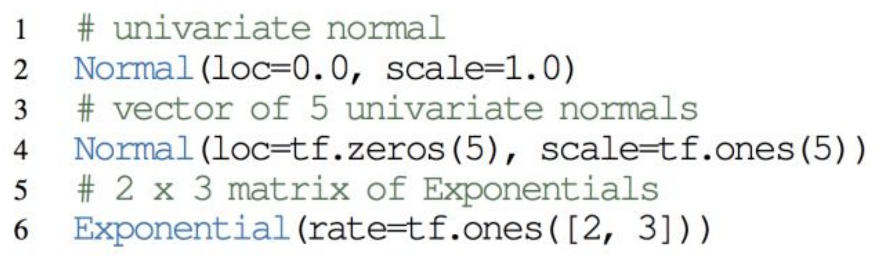
[Tran et al, 2017]

Models

[Tran's slide]

Edward's language augments computational graphs with an abstraction for random variables

A *random variable x* is an object parametrized by *tensors* 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 explicit methods such as log_prob( ) and sample( )

Each random variable is associated to a tensor $\mathbf{x}^*,\ \mathbf{x}^* \sim p(\mathbf{x} \mid \theta^*)$

For implementation, all TensorFlow Distributions and call sample are wrapped to produce the associated tensor.

[Tran et al, 2017]

[Rowan’s slide]

2-Layer Neural Network in TensorFlow

```
import tensorflow as tf

def neural_network(x, W_0, W_1, W_2, b_0, b_1, b2):
  h = tf.tanh(tf.matmul(x, W_0) + b_0)
  h = tf.tanh(tf.matmul(h, W_1) + b_1)
  h = tf.matmul(h, W_2) + b_2
  return tf.reshape(h, [-1])
```

[Rowan's slide]

2-Layer Neural Network in TensorFlow + Edward

```
from edward.models import Normal

W_0 = Normal(loc=tf.zeros([D, n_hidden]), scale=tf.ones([D, n_hidden]))
W_1 = Normal(loc=tf.zeros([n_hidden, n_hidden]), scale=tf.ones([n_hidden, n_hidden]))
W_2 = Normal(loc=tf.zeros([n_hidden, 1]), scale=tf.ones([n_hidden, 1]))
b_0 = Normal(loc=tf.zeros(n_hidden), scale=tf.ones(n_hidden))
b_1 = Normal(loc=tf.zeros(n_hidden), scale=tf.ones(n_hidden))
b_2 = Normal(loc=tf.zeros(1), scale=tf.ones(1))
```

Eg: Beta-Bernoulli

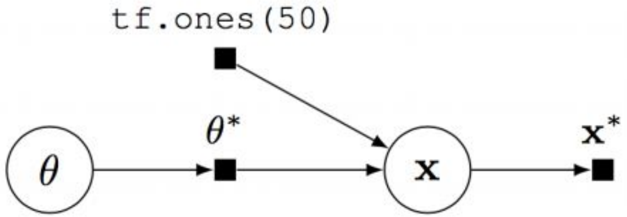
[Tran's slide]

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

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



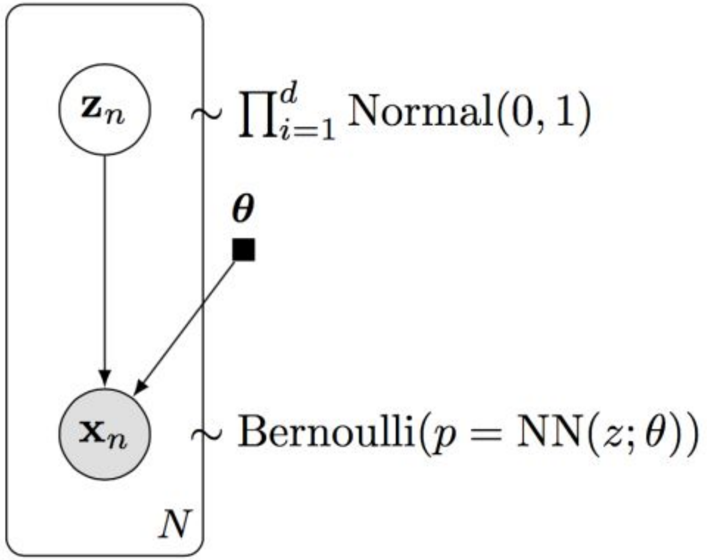
Fetching **x** from the graph generates a binary vector of 50 elements.

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

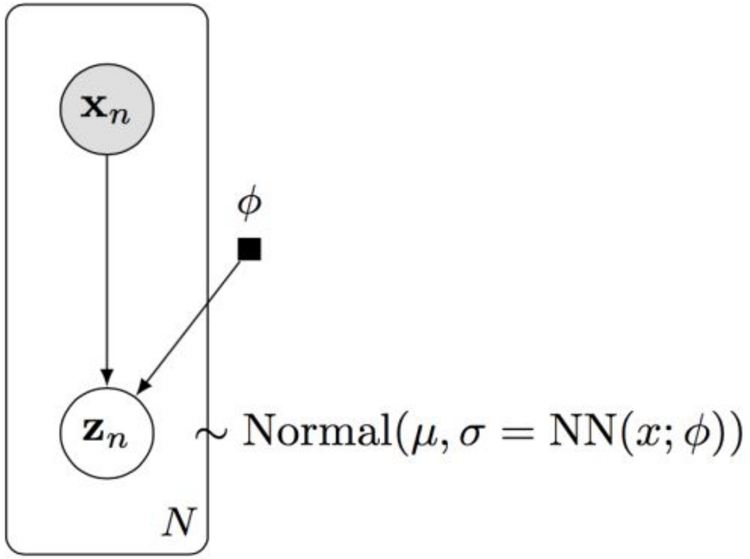
[Tran et al, 2017]

Eg: Variational Auto-Encoder for Binarized MNIST

[Tran's slide]



$\mathbf{z}_n$
$\sim \prod_{i=1}^{d} \text{Normal}(0,1)$
$\boldsymbol{\theta}$
$\mathbf{x}_n$
$\sim \text{Bernoulli}(p = \text{NN}(z;\theta))$
N



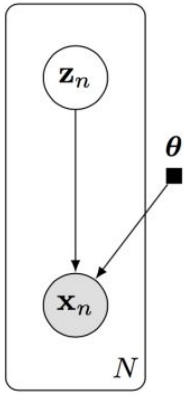
Decoder

Encoder

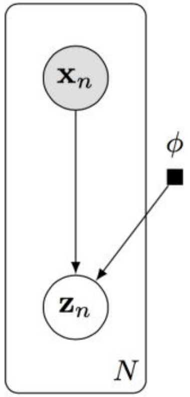
[Tran et al, 2017]

Eg: Variational Auto-Encoder for Binarized MNIST

[Tran's slide]



```
# 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))
```

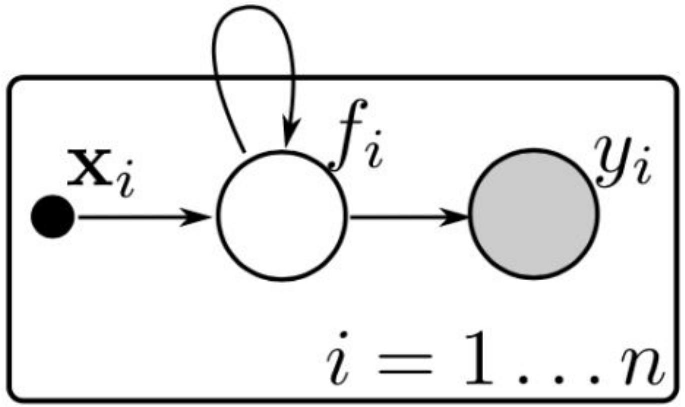


```
# Variational model
qx = tf.placeholder(tf.float32, [N, 28 * 28])
qh = Dense(256, activation='relu')(qx)
qz = Normal(loc=Dense(d, activation=None)(qh),
            scale=Dense(d, activation='softplus')(qh))
```

[Tran et al, 2017]

Eg: Gaussian Process Classification

[Tran's slide]

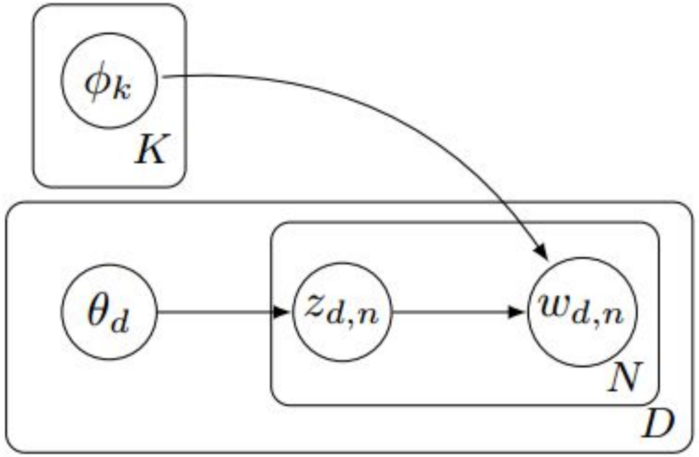


```
X = tf.placeholder(tf.float32, [N, D])
f = MultivariateNormalTriL(loc=tf.zeros(N),
                           scale_tril=tf.cholesky(rbf(X)))
y = Bernoulli(logits=f)
```

[Tran et al, 2017]

Eg: Latent Dirichlet Allocation

[Tran's slide]



```
D = 4  # number of documents
N = [11502, 213, 1523, 1351]  # words per doc
K = 10  # number of topics
V = 100000  # vocabulary size

theta = Dirichlet(alpha=tf.zeros([D, K]) + 0.1)
phi = Dirichlet(alpha=tf.zeros([K, V]) + 0.05)
z = [[0] * N] * D
w = [[0] * N] * D
for d in range(D):
  for n in range(N[d]):
    z[d][n] = Categorical(pi=theta[d, :])
    w[d][n] = Categorical(pi=phi[z[d][n], :])
```

[Tran et al, 2017]

Inference

What is inference in Edward?

- A key concept in Edward is that there is no distinct “model” or “inference” block. **A model is simply a collection of random variables, and inference is a way of modifying parameters in that collection subject to another.**

- Flexibility

- For example, we can infer only parts of a model (e.g., layer-wise training), infer parts used in multiple models (e.g., multi-task learning), or plug in a posterior into a new model (e.g., Bayesian updating)

Inference

[Tran's slide]

Given

- Data $\mathbf{x}_{\text{train}}$.
- Model $p(\mathbf{x}, \mathbf{z}, \beta)$ of observed variables $\mathbf{x}$ and latent variables $\mathbf{z}, \beta$.

Goal

- Calculate posterior distribution

$$p(\mathbf{z}, \beta \mid \mathbf{x}_{\text{train}}) = \frac{p(\mathbf{x}_{\text{train}}, \mathbf{z}, \beta)}{\int p(\mathbf{x}_{\text{train}}, \mathbf{z}, \beta) \, \mathrm{d}\mathbf{z} \, \mathrm{d}\beta}.$$

This is the key problem in Bayesian inference.

What is posterior distribution?

What is posterior distribution?

- The posterior distribution is a way to summarize what we know about uncertain quantities in Bayesian analysis. It is a combination of the prior distribution and the likelihood function, which tells you what information is contained in your observed data (the “new evidence”).

- In other words, the posterior distribution summarizes what you know after the data has been observed. The summary of the evidence from the new observations is the likelihood function.

Posterior Distribution = Prior Distribution + Likelihood Function (“new evidence”)

Example for posterior distribution calculation

Pregnancy Test Example from Basics of Bayesian Statistics:

Given:

- Accuracy rate of pregnancy tests = 0.9
- False positive results = 50%

Thus, there are two possible events **Bi: B1 = preg and B2 = not preg.**
Given the accuracy and false-positive rates, we know the conditional probabilities of obtaining a positive test under these events:

- p(test +|preg) = .9 and p(test +|not preg) = .5.

$$p(\text{preg} \mid \text{test} +) = \frac{p(\text{test} + \mid \text{preg})p(\text{preg})}{p(\text{test} + \mid \text{preg})p(\text{preg}) + p(\text{test} + \mid \text{not preg})p(\text{not preg})}.$$

Filling in the known information yields:

$$p(\text{preg} \mid \text{test} +) = \frac{(.90)(.15)}{(.90)(.15) + (.50)(.85)} = \frac{.135}{.135 + .425} = .241.$$

This probability is called as the posterior probability

Issue of Concern

- If the woman is aware of the test’s limitations, she may choose to repeat the test. Now, she can use the “updated” probability of being pregnant (p = .241)

- This result is still not very convincing evidence that she is pregnant, but if she repeats the test again and finds a positive result, her probability increases to .364 (for general interest, subsequent positive tests yield the following probabilities: test 3= .507, test 4 = .649, test 5 = .769, test 6 = .857, test 7 = .915, test 8 = .951, test 9 = .972, test 10 = .984).

- This process of repeating the test and recomputing the probability of interest is the basic process of concern in Bayesian statistics.

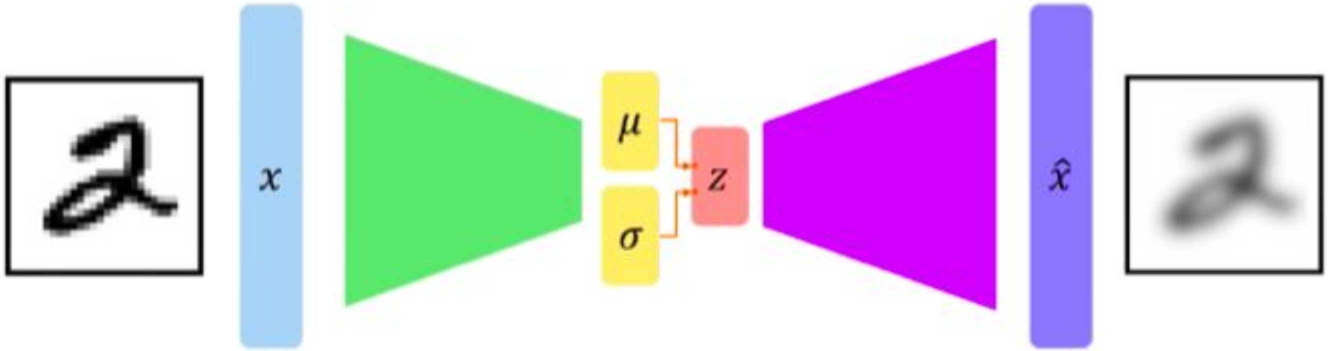
Classes of Inference

- Variational Inference: Variational inference posits a family of approximating distributions and finds the closest member in the family to the posterior.

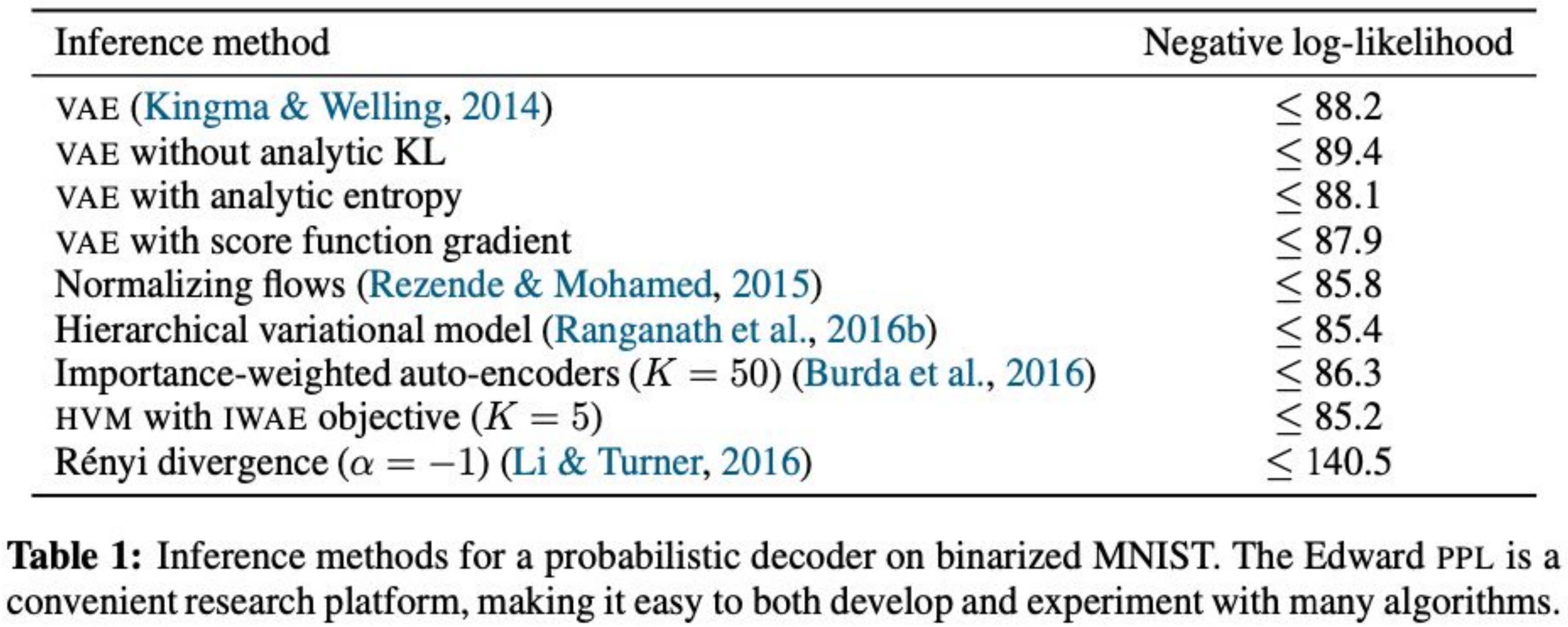
- Monte Carlo: It utilizes the MAP (Maximum a posteriori) in order to perform estimation with an approximating family (qbeta and qz) of PointMass random variables, i.e., with all probability mass concentrated at a point.

- Non-Bayesian Methods: The model posits random noise eps over N data points, each with d dimensions; this random noise feeds into a generative_network function, a neural network that outputs real-valued data x. In addition, there is a discriminative_network which takes data as input and outputs the probability that the data is real (in logit parameterization). We build GANInference; running it optimizes parameters inside the two neural network functions. This approach extends to many advances in GANs

What is a Variational Autoencoder?



Variational Inference



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 is a convenient research platform, making it easy to both develop and experiment with many algorithms.

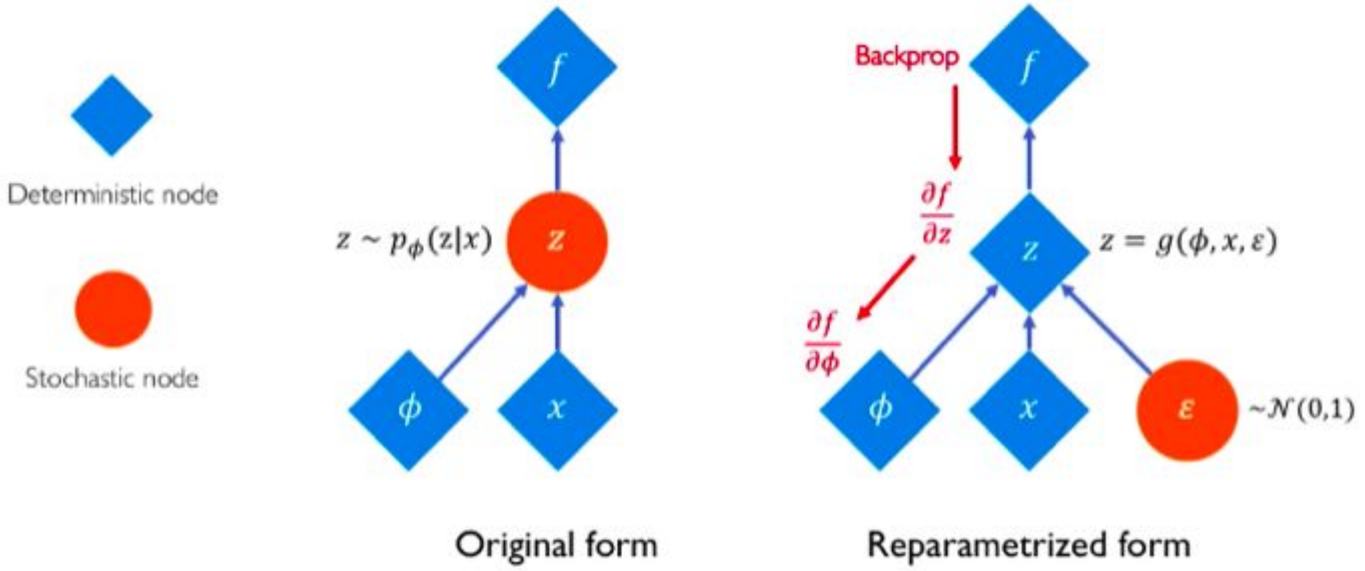
Variational Inference

- The first one represents the regular VAE as discussed earlier (VAE with MNIST)
- The second,third and fourth use the same VAE as in the first one but only different gradient estimators.
- This means they reach the same optima but at different convergence rates
- The authors further even use Hierarchical Variational Models (HVMs) with a normalized flow prior.
- This results in better results than Importance Weighted Autoencoders (IWAEs).

Variants of VAE used

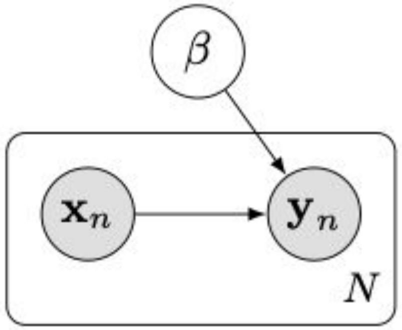
- VAE reparameterization without analytic KL
- VAE reparameterization with analytic entropy
- VAE with score function gradient

Best performing VAE for Variational Inference



VAE network with and without the "reparameterization" trick (Source)

GPU-ACCELERATED HAMILTONIAN MONTE CARLO



```
# Model
x = tf.Variable(x_data, trainable=False)
beta = Normal(mu=tf.zeros(D), sigma=tf.ones(D))
y = Bernoulli(logits=tf.dot(x, beta))

# Inference
qbeta = Empirical(params=tf.Variable(tf.zeros([T, D])))
inference = ed.HMC({beta: qbeta}, data={y: y_data})
inference.run(step_size=0.5 / N, n_steps=10)
```

Figure 9: Edward program for Bayesian logistic regression with Hamiltonian Monte Carlo (HMC).

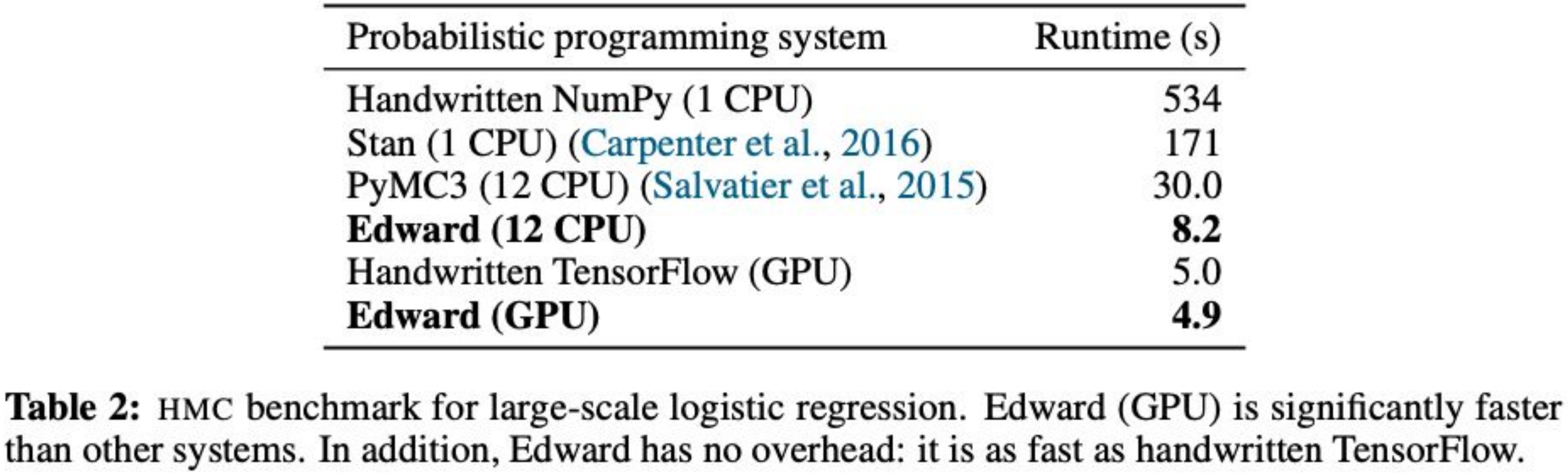
What is Hamiltonian Monte Carlo technique?

- A Hamiltonian Monte Carlo (HMC) sampler is a gradient-based Markov Chain Monte Carlo sampler that you can use to generate samples from a probability density *P(x)*.

- HMC sampling requires specification of log P(x) and its gradient.The parameter vector *x* must be unconstrained, meaning that every element of *x* can be any real number.

- After creating a sampler, you can compute MAP (maximum-a-posteriori) point estimates, tune the sampler, draw samples, and check convergence diagnostics using the methods of this class.

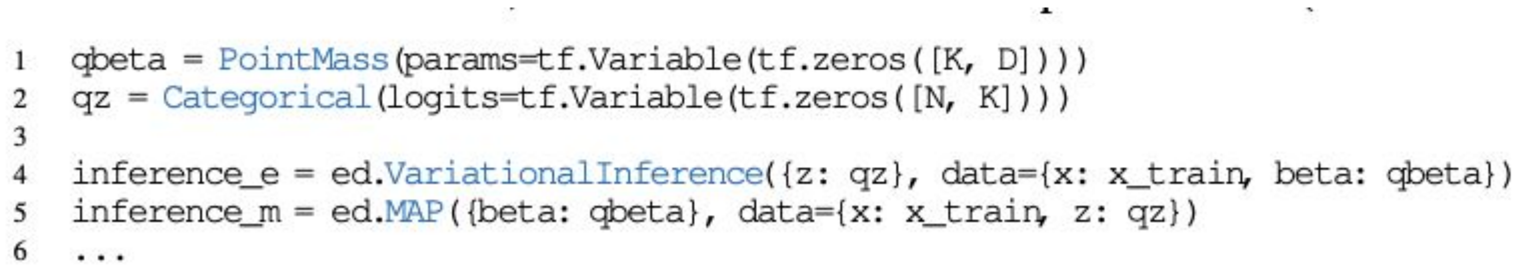
Results for HMC



Probabilistic programming system	Runtime (s)
Handwritten NumPy (1 CPU)	534
Stan (1 CPU) (Carpenter et al., 2016)	171
PyMC3 (12 CPU) (Salvatier et al., 2015)	30.0
Edward (12 CPU)	**8.2**
Handwritten TensorFlow (GPU)	5.0
Edward (GPU)	**4.9**

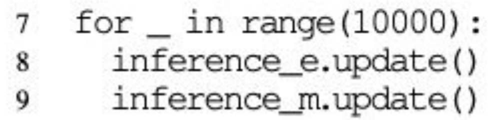
Table 2: HMC benchmark for large-scale logistic regression. Edward (GPU) is significantly faster than other systems. In addition, Edward has no overhead: it is as fast as handwritten TensorFlow.

Compositional Inference



```
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_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()
```

Conclusion

Goal: "As flexible and computationally efficient as traditional DL"

- Variety of composable inference methods
- Recent advances in variational inference methods
- GANs
- Inference algorithms

- Computational graphs
- Scales to massive data
- No runtime overhead

[Tran et al, 2017]

Future Work & Open Challenges

1. Better facilitate programs with complex control flow and recursion

1. Expand Edward's design to dynamic computational graphs frameworks

[Tran et al, 2017]

THANK YOU !!!