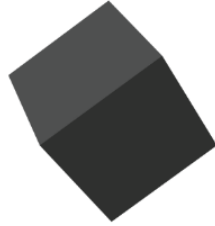


Edward

Edward




Getting Started

Tutorials

API

Community

Contributing

Github 

A library for probabilistic modeling, inference, and criticism.

Edward is a Python library for probabilistic modeling, inference, and criticism. It is a testbed for fast experimentation and research with probabilistic models, ranging from classical hierarchical models on small data sets to complex deep probabilistic models on large data sets. Edward fuses three fields: Bayesian statistics and machine learning, deep learning, and probabilistic programming.

It supports **modeling** with

- Directed graphical models
- Neural networks (via libraries such as [Keras](#) and [TensorFlow Slim](#))
- Implicit generative models
- Bayesian nonparametrics and probabilistic programs

It supports **inference** with

- Variational inference
 - Black box variational inference
 - Stochastic variational inference
 - Generative adversarial networks
 - Maximum a posteriori estimation
- Monte Carlo
 - Gibbs sampling
 - Hamiltonian Monte Carlo
 - Stochastic gradient Langevin dynamics
- Compositions of inference
 - Expectation-Maximization
 - Pseudo-marginal and ABC methods
 - Message passing algorithms

It supports **criticism** of the model and inference with

Contents

- Introduction
- Edward: A library for probabilistic modeling, inference, and criticism
- Deep Probabilistic Programming (Related to Deep Generative Model)

Introduction

- Edward supports a broad class of probabilistic models, efficient algorithms for inference, and many techniques for model criticism
 - Named after the statistician George Edward Pelham Box
- For modeling
 - Directed graphical models
 - Stochastic neural networks

- Programs with stochastic control flow (?)
- For inference, Edward provides algorithms
 - **Stochastic and black box variational inference**
 - Hamiltonian Monte Carlo
 - Stochastic gradient Langevin dynamics
 - Infrastructure to make it easy to develop new algorithms.
- For criticism,
 - Scoring rules
 - Predictive checks
- Built on top of **TensorFlow** to support distributed training and hardware such as GPUs

Why Edward?

We are interested in deploying probabilistic models to many real world applications, ranging from the size of data and data structure, such as large text corpora or many brief audio signals, to the size of model and class of models, such as small nonparametric models or "DEEP GENERATIVE MODELS".

- Various Inference
- Computational Frameworks
- High-level Deep Learning Libraries
 - Similar to Keras
 - We can use Keras and Edward together.

References

- Edward: A library for probabilistic modeling, inference, and criticism (2016)
 - Youtube Video
 - <https://youtu.be/PvyVahNI8H8> (<https://youtu.be/PvyVahNI8H8>)
 - <https://youtu.be/4XZkHtHtQsk> (<https://youtu.be/4XZkHtHtQsk>)
- DEEP PROBABILISTIC PROGRAMMING (2017)

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

SimpleExample

....

Model & Composing Random Variables

Directed Graphical Model

```

1 from edward.models import Bernoulli, Beta
2
3 theta = Beta(a=1.0, b=1.0)
4 x = Bernoulli(p=tf.ones(50) * theta)

```

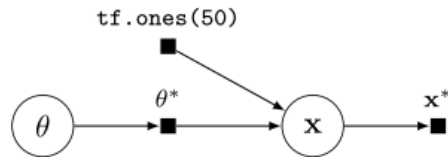


Figure 5: Computational graph for a Beta-Bernoulli program.

Neural Network

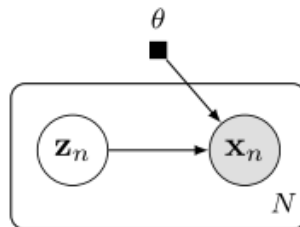


Figure 6: Graphical representation of a deep generative model.

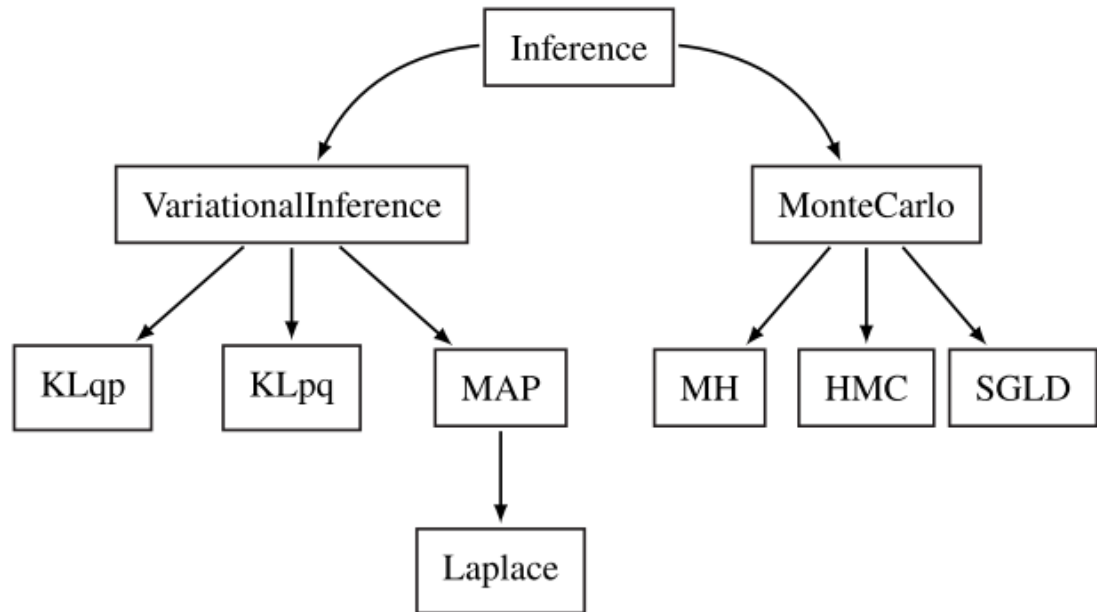
With TensorFlow Slim, we write this model as follows:

```

1 from edward.models import Bernoulli, Normal
2 from tensorflow.contrib import slim
3
4 z = Normal(mu=tf.zeros([N, d]), sigma=tf.ones([N, d]))
5 h = slim.fully_connected(z, 256)
6 x = Bernoulli(logits=slim.fully_connected(h, 28 * 28, activation_fn=None))

```

Inference



Composing Random Variables

Hybrid Algorithms

```

1 from edward.models import Categorical, PointMass
2
3 qbeta = PointMass(params=tf.Variable(tf.zeros([K, D])))
4 qz = Categorical(logits=tf.Variable(tf.zeros[N, K]))
5
6 inference_e = ed.VariationalInference({z: qz}, data={x: x_data, beta: qbeta})
7 inference_m = ed.MAP({beta: qbeta}, data={x: x_data, z: qz})
8 ...
9 for _ in range(10000):
10     inference_e.update()
11     inference_m.update()

```

Message Passing

```

1 from edward.models import Categorical, Normal
2
3 N1 = 1000 # number of data points in first data set
4 N2 = 2000 # number of data points in second data set
5 D = 2 # data dimension
6 K = 5 # number of clusters
7
8 # MODEL
9 beta = Normal(mu=tf.zeros([K, D]), sigma=tf.ones([K, D]))
10 z1 = Categorical(logits=tf.zeros([N1, K]))
11 z2 = Categorical(logits=tf.zeros([N2, K]))
12 x1 = Normal(mu=tf.gather(beta, z1), sigma=tf.ones([N1, D]))
13 x2 = Normal(mu=tf.gather(beta, z2), sigma=tf.ones([N2, D]))
14
15 # INFERENCE
16 qbeta = Normal(mu=tf.Variable(tf.zeros([K, D])),
17                sigma=tf.nn.softplus(tf.Variable(tf.zeros([K, D]))))
18 qz1 = Categorical(logits=tf.Variable(tf.zeros[N1, K]))
19 qz2 = Categorical(logits=tf.Variable(tf.zeros[N2, K]))
20
21 inference_z1 = ed.KLpq({beta: qbeta, z1: qz1}, {x1: x1_train})
22 inference_z2 = ed.KLpq({beta: qbeta, z2: qz2}, {x2: x2_train})
23 ...
24 for _ in range(10000):
25     inference_z1.update()
26     inference_z2.update()

```

Deep Probabilistic Programming

- Compositionality
 - The nature of deep neural networks is compositional.
 - Users can connect layers in creative ways, without having to worry about how to perform testing or inference.
 - testing (forward propagation)
 - inference (gradient- based optimization, with back propagation and automatic differentiation)
- Compositional representations for probabilistic programming
 - Random variables
 - Inference
- Probabilistic programming lets users
 - Specify generative probabilistic models as programs
 - “compile” those models down into inference procedures.
- We propose Edward, a new Turing-complete probabilistic programming language
 - Builds on two compositional representations (random variables and inference).
- Aim to design compositional representations for probabilistic programming.
 - Has focused on how to build rich probabilistic programs by composing random variables
 - Such systems cannot capture recent advances in probabilistic modeling such as in variational inference
 - Analogous compositionality for inference.

VAE

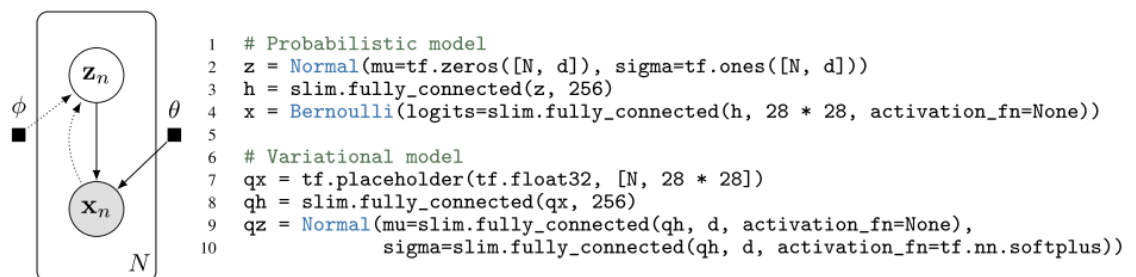
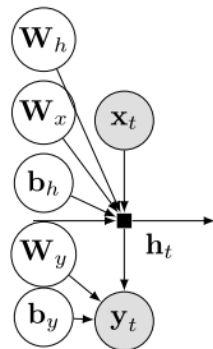


Figure 1: Variational auto-encoder for a data set of 28×28 pixel images: (left) graphical model, with dotted lines for the inference model; (right) probabilistic program, with 2-layer neural networks.

Bayesian RNN



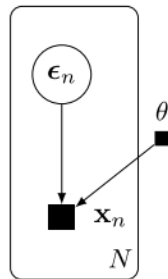
```

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)

```

Figure 3: Bayesian RNN: **(left)** graphical model; **(right)** probabilistic program. The program has an unspecified number of time steps; it uses a symbolic for loop (`tf.scan`).

GAN



```

1 def generative_network(eps):
2     h = Dense(256, activation='relu')(eps)
3     return Dense(28 * 28, activation=None)(h)
4
5 def discriminative_network(x):
6     h = Dense(28 * 28, activation='relu')(x)
7     return Dense(h, activation=None)(1)
8
9 # Probabilistic model
10 eps = Normal(mu=tf.zeros([N, d]), sigma=tf.ones([N, d]))
11 x = generative_network(eps)
12
13 inference = ed.GANInference(data={x: x_train},
14                             discriminator=discriminative_network)

```

Figure 7: Generative adversarial networks: **(left)** graphical model; **(right)** probabilistic program. The model (generator) uses a parameterized function (discriminator) for training.

In []:

1