Deep Probabilistic Programming with Edward

Dustin Tran†, Matt Hoffman*‡, Kevin Murphy‡, Eugene Brevdo‡, Rif Saurous‡, David Blei†

†Columbia University, *Adobe Research, ‡Google

edwardlib.org/iclr2017

Summary

- Deep neural networks are popular in large part due to their compositional nature. How do we do this for probabilistic modeling?
- We describe Edward, a Turing-complete probabilistic programming language.
- Edward builds two representations—random variables and inference.
- For example, we show how to design rich variational models and generative adversarial networks.

Compositional Representations for Probabilistic Models

- We define random variables as the key compositional representation.
- They are class objects e.g. with log-density and sample methods.
- Each random variable $\mathbf{x}$ is associated to a tensor $\mathbf{x}^*$ in the computational graph, which represents a single sample $\mathbf{x}^* \sim p(\mathbf{x})$.
- Mutable states represent enable conditioning sets to vary, $p(\mathbf{y} \mid \mathbf{x})$ and optimization of parameters, $p(\mathbf{x}; \theta)$.

Compositional Representations for Inference

- Given data $\mathbf{x}_{\text{train}}$, inference aims to calculate the posterior $p(\mathbf{z}, \beta \mid \mathbf{x}_{\text{train}}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are any model parameters to estimate.
- In variational inference, the idea is to posit an approximating family $q \in \mathcal{Q}$ and to find the closest member q^*. We write it with mutable states representing its parameters, where $q(\beta; \mu, \sigma) = \text{Normal}(\beta; \mu, \sigma)$, $q(\mathbf{z}; \pi) = \text{Categorical}(\mathbf{z}; \pi)$.

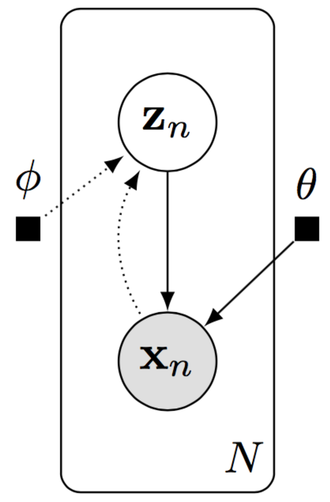
```
qbeta = Normal(mu=tf.Variable(tf.zeros([K, D])),
               sigma=tf.exp(tf.Variable(tf.zeros[K, D])))
qz = Categorical(logits=tf.Variable(tf.zeros[N, K]))

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

- Specific variational algorithms inherit from VariationalInference to define their own methods, e.g., a loss function and gradient.
- Monte Carlo approximates the posterior using samples. We represent it where the approximating family is an empirical distribution, $q(\beta; \{\beta^{(t)}\}) = \frac{1}{T}\sum_{t=1}^{T} \delta(\beta, \beta^{(t)})$, $q(\mathbf{z}; \{\mathbf{z}^{(t)}\}) = \frac{1}{T}\sum_{t=1}^{T} \delta(\mathbf{z}, \mathbf{z}^{(t)})$.

```
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})
```

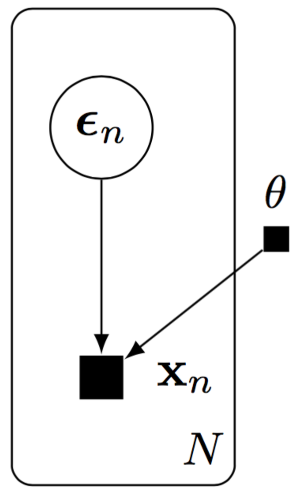
- Monte Carlo algorithms proceed by updating one sample $\beta^{(t)}, \mathbf{z}^{(t)}$ at a time in the empirical approximation. Specific MC samplers determine the update rules.

Example: Variational Auto-Encoder



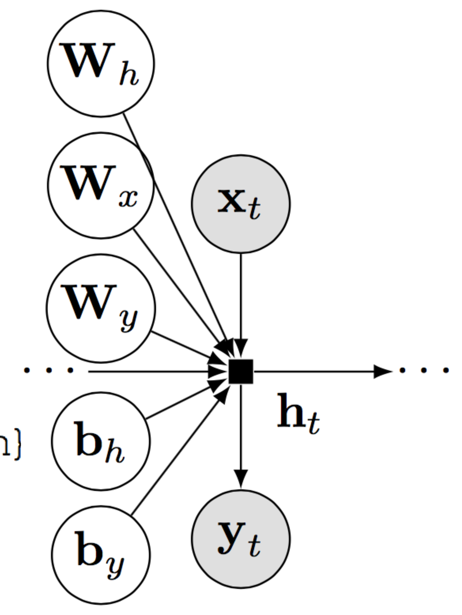

```
# Probabilistic model
z = Normal(mu=tf.zeros([N, d]), sigma=tf.ones([N, d]))
h = Dense(256, activation='relu')(z)
x = Bernoulli(logits=Dense(28 * 28, activation=None)(h))

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

Example: Generative Adversarial Networks




```
def generative_network(eps):
  h = Dense(256, activation='relu')(eps)
  return Dense(28 * 28, activation=None)(h)

def discriminative_network(x):
  h = Dense(28 * 28, activation='relu')(x)
  return Dense(h, activation=None)(1)

# Probabilistic model
eps = Normal(mu=tf.zeros([M, d]), sigma=tf.ones([M, d]))
x = generative_network(eps)

inference = ed.GANInference(data={x: x_train},
    discriminator=discriminative_network)
```

Example: Bayesian RNN with Variable Length




```
def rnn_cell(hprev, xt):
  return tf.tanh(tf.dot(hprev, Wh) + tf.dot(xt, Wx) + bh)

Wh = Normal(mu=tf.zeros([H, H]), sigma=tf.ones([H, H]))
Wx = Normal(mu=tf.zeros([D, H]), sigma=tf.ones([D, H]))
Wy = Normal(mu=tf.zeros([H, 1]), sigma=tf.ones([H, 1]))
bh = Normal(mu=tf.zeros(H), sigma=tf.ones(H))
by = Normal(mu=tf.zeros(1), sigma=tf.ones(1))

x = tf.placeholder(tf.float32, [None, D])
h = tf.scan(rnn_cell, x, initializer=tf.zeros(H))
y = Normal(mu=tf.matmul(h, Wy) + by, sigma=1.0)
```

Composing Inferences

Core to Edward's design is that inference can be written as a collection of separate inference programs. Below we demonstrate variational EM.

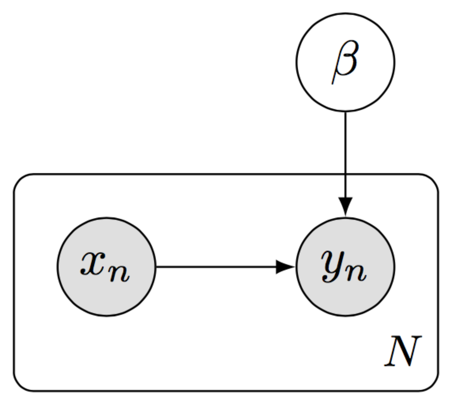
```
qbeta = PointMass(params=tf.Variable(tf.zeros([K, D])))
qz = Categorical(logits=tf.Variable(tf.zeros[N, K]))

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()
```

Experiments: Recent Methods in Variational Inference

Inference method	Negative log-likelihood
Variational auto-encoder (VAE) [2]	$\leq$ 88.2
VAE without analytic KL	$\leq$ 89.4
VAE with analytic entropy	$\leq$ 88.1
VAE with score function gradient	$\leq$ 87.9
Normalizing flows [4]	$\leq$ 85.8
Hierarchical variational model [3]	$\leq$ 85.4
Importance-weighted auto-encoders ($K = 50$) [1]	$\leq$ 86.3
HVM with IWAE objective ($K = 5$)	$\leq$ 85.2
Rényi divergence ($\alpha = -1$)	$\leq$ 140.5

Inference methods for a probabilistic decoder on binarized MNIST. The Edward PPL enables fast experimentation with many algorithms.

Experiments: 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=100)
```

We apply Bayesian logistic regression to Covertype ($N = 581012$, $D = 54$). 12-core Intel i7-5930K CPU at 3.50GHz, a NVIDIA Titan X (Maxwell) GPU. We compare the runtime of HMC for 100 iterations (and same settings).

Probabilistic programming system	Runtime (s)
Handwritten NumPy (1 CPU)	534
Stan (1 CPU)	171
PyMC3 (12 CPU)	30.0
Edward (12 CPU)	**8.2**
Handwritten TensorFlow (GPU)	5.0
Edward (GPU)	**4.9** (35x faster than Stan)

Edward (GPU) is significantly faster than other systems. In addition, Edward has no overhead: it is as fast as handwritten TensorFlow.

References

[1] Burda, Y., Grosse, R., and Salakhutdinov, R. (2016). Importance weighted autoencoders. In *International Conference on Learning Representations*.

[2] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. In *International Conference on Learning Representations*.

[3] Ranganath, R., Tran, D., and Blei, D. M. (2016). Hierarchical variational models. In *International Conference on Machine Learning*.

[4] Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. In *International Conference on Machine Learning*.