Edward: A Library for Probabilistic Modeling



Dustin Tran[†], Matt Hoffman^{*‡}, Kevin Murphy[‡], Eugene Brevdo[‡], Rif Saurous[‡], David Blei[†]

[†]Columbia University, ^{*}Adobe Research, [‡]Google edwardlib.org



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 new 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 **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, $p(\mathbf{y}|\mathbf{x})$ and optimization of parameters, $p(\mathbf{x};\theta)$.

Compositional Representations for Inference

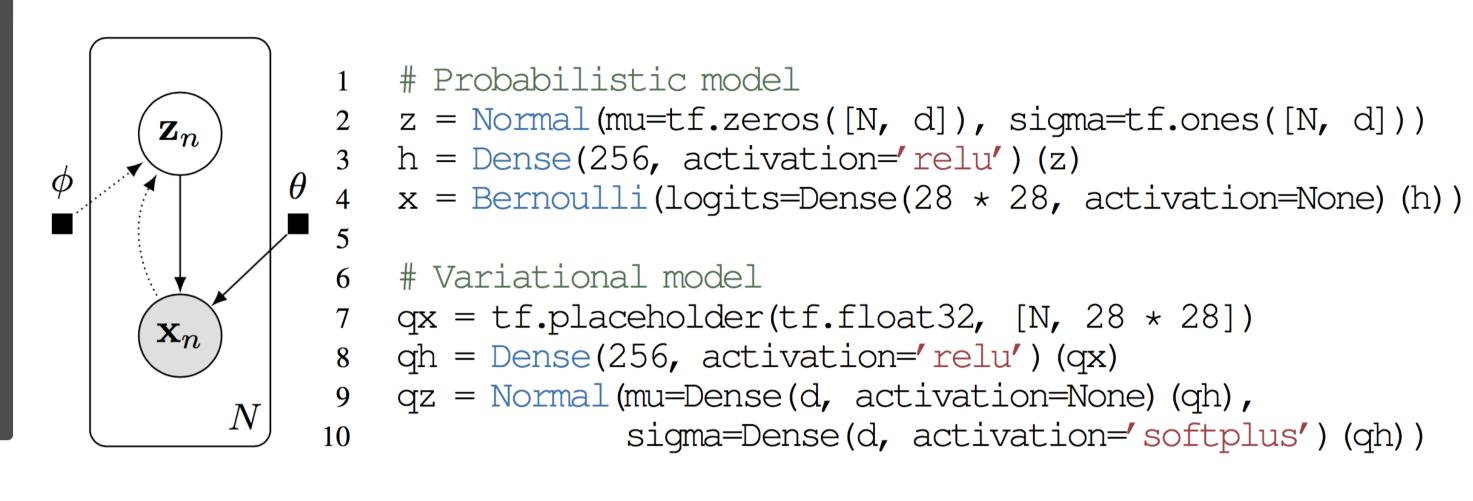
- Given data $\mathbf{x}_{\text{train}}$, inference aims to calculate the posterior $p(\mathbf{z}, \beta \mid \mathbf{x}_{\text{train}}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are any model parameters to estimate.
- In variational inference, the idea is to posit an approximating family $q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

- 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, $q(\beta; \{\beta^{(t)}\}) = \frac{1}{T} \sum_{t=1}^{T} \delta(\beta, \beta^{(t)}), q(\mathbf{z}; \{\mathbf{z}^{(t)}\}) = \frac{1}{T} \sum_{t=1}^{T} \delta(\mathbf{z}, \mathbf{z}^{(t)}).$

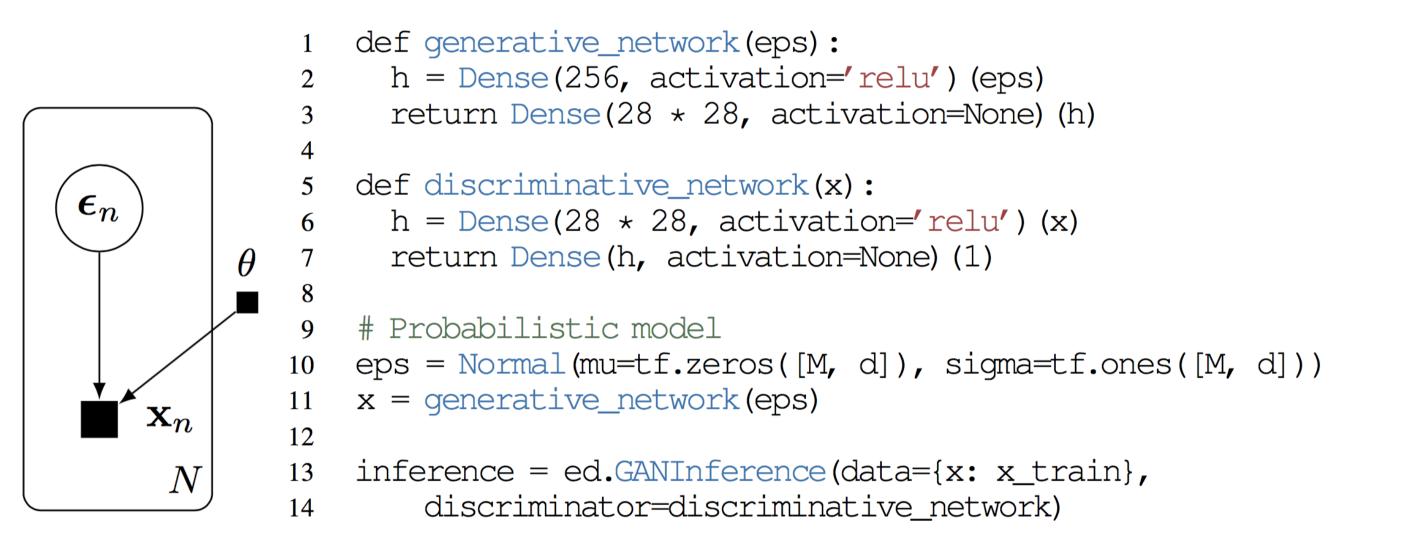
```
1  T = 10000 # number of samples
2  qbeta = Empirical(params=tf.Variable(tf.zeros([T, K, D]))
3  qz = Empirical(params=tf.Variable(tf.zeros([T, N]))
4
5  inference = ed.MonteCarlo({beta: qbeta, z: qz}, data={x: x_train})
```

• 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.

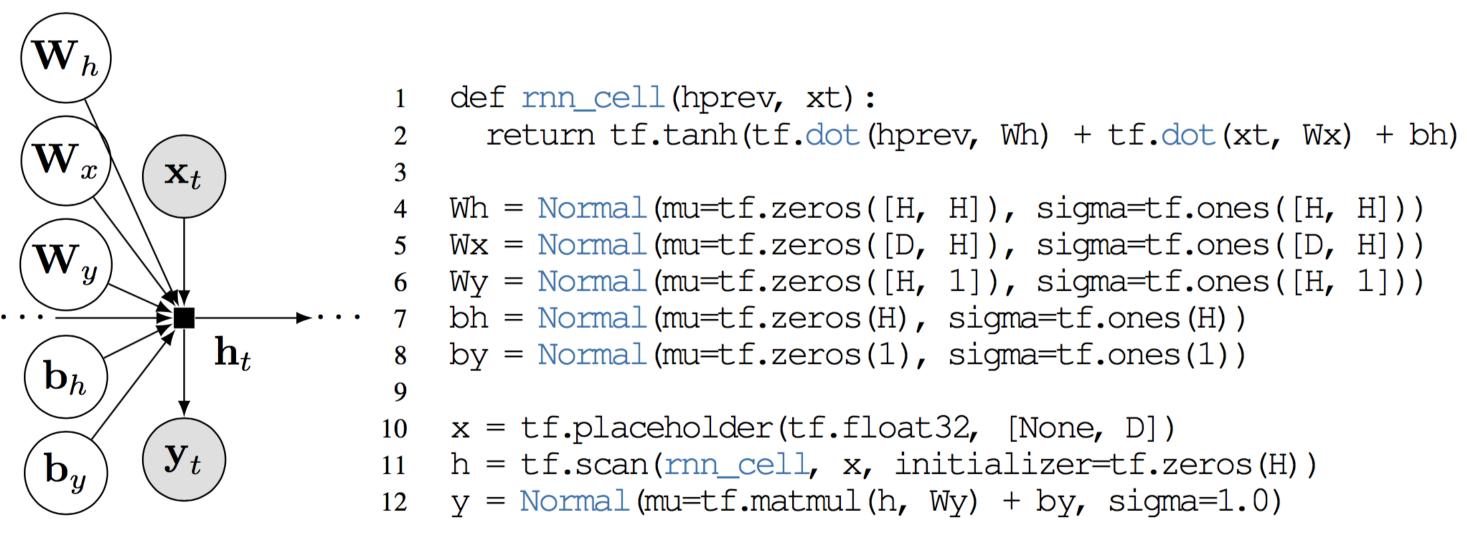
Example: Variational Auto-Encoder



Example: Generative Adversarial Networks



Example: Bayesian RNN with Variable Length



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})

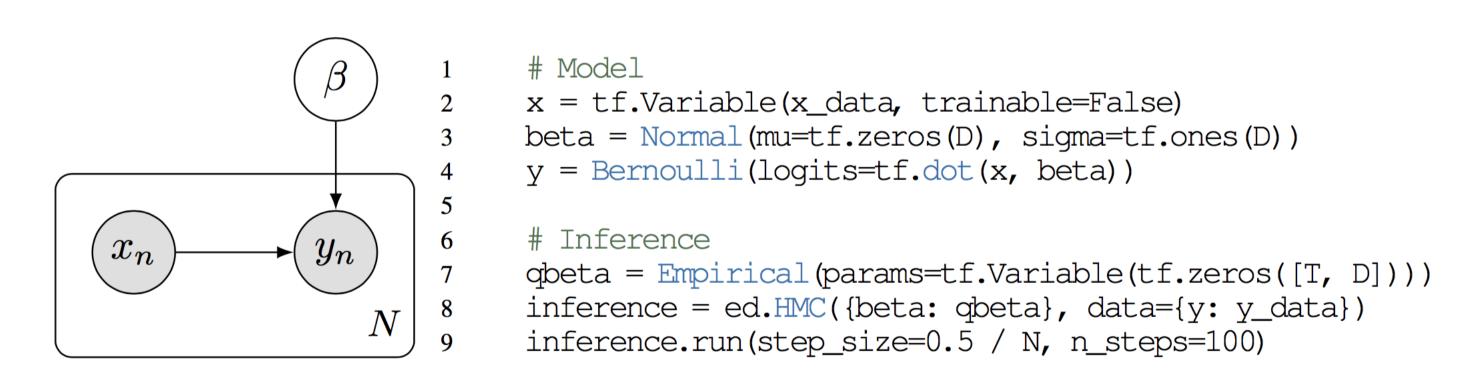
for _ in range(10000):
    inference_e.update()
    inference_m.update()
```

Experiments: Recent Methods in Variational Inference

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

Inference methods for a probabilistic decoder on binarized MNIST. The Edward PPL makes it easy to experiment with many algorithms.

Experiments: GPU-accelerated Hamiltonian Monte Carlo



We perform inference on Bayesian logistic regression for the Covertype dataset (N = 581012, D = 54). We use a 12-core Intel i7-5930K CPU at 3.50GHz and a NVIDIA Titan X (Maxwell) GPU.

We compare the runtime of HMC for 100 iterations (and same settings).

Probabilistic programming language Runtime	
Stan (1 CPU) [2]	171 sec
PyMC3 (12 CPU) [6]	361 sec
Edward (12 CPU)	8.2 sec
Edward (GPU)	4.9 sec (35x faster than Stan)

References

- Burda, Y., Grosse, R., and Salakhutdinov, R. (2016). Importance weighted autoencoders. In *International Conference on Learning Representations*.
- [2] Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2016). Stan: A probabilistic programming language. *Journal of Statistical Software*.
- Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. In *International Conference on Learning Representations*.
- Ranganath, R., Tran, D., and Blei, D. M. (2016). Hierarchical variational models. In *International Conference on Machine Learning*.
- [5] Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. In *International Conference on Machine Learning*.
- [6] Salvatier, J., Wiecki, T., and Fonnesbeck, C. (2015). Probabilistic Programming in Python using PyMC. *arXiv* preprint *arXiv*:1507.08050.