Edward: Deep Probabilistic Programming

Extended Seminar – Systems and Machine Learning

Steven Lang

13.02.2020

Outline

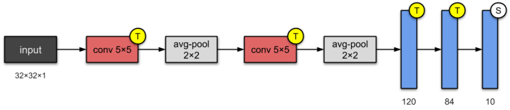
Outline

Motivation

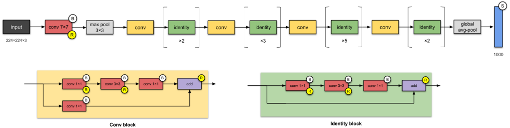
- Nature of deep neural networks is **compositional**
- Connect layers in creative ways
- No worries about
  - testing (forward propagation)
  - inference (gradient based opt., with backprop. and auto-diff.)
- Leads to easy development of new successful architectures

Motivation

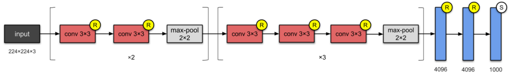
LeNet-5 (Lecun et al. 1998)



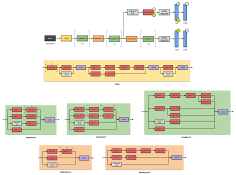
ResNet-50 (He et al. 2015)



VGG16 (Simonyan and Zisserman 2014)



Inception-v4 (Szegedy et al. 2014)



Motivation

Goal: Achieve the composability of deep learning for

1. Probabilistic models
2. Probabilistic inference

Outline

What is a Random Variable (RV)?

- Random number determined by chance, e.g. outcome of a single dice roll
- Drawn according to a probability distribution
- Typical random variables in statistical machine learning:
  - input data
  - output data
  - noise

What is a Probability Distribution?

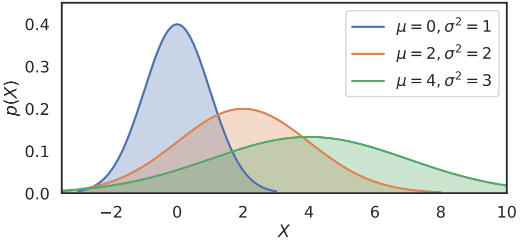
- **Discrete**: Describes probability, that RV will be equal to a certain value
- **Continuous**: Describes probability *density*, that RV will be equal to a certain value

What is a Probability Distribution?

- **Discrete**: Describes probability, that RV will be equal to a certain value
- **Continuous**: Describes probability *density*, that RV will be equal to a certain value

Example: Normal distribution

$$\mathcal{N}(\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$



Common Probability Distributions

Discrete

- Bernoulli
- Binomial
- Hypergeometric
- Poisson
- Boltzmann

Common Probability Distributions

Discrete

- Bernoulli
- Binomial
- Hypergeometric
- Poisson
- Boltzmann

Continuous

- Uniform
- Beta
- Normal
- Laplace
- Student-t

What is Inference?

- Answer the query $P(\mathbf{Q} \mid \mathbf{E})$
  - $\mathbf{Q}$: Query, set of RVs we are interested in
  - $\mathbf{E}$: Evidence, set of RVs that we know the state of

What is Inference?

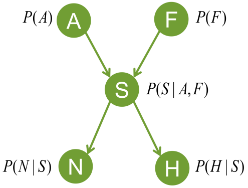
- Answer the query $P(\mathbf{Q} \mid \mathbf{E})$
  - $\mathbf{Q}$: Query, set of RVs we are interested in
  - $\mathbf{E}$: Evidence, set of RVs that we know the state of

- Example: What is the prob. that
  - it has rained ($\mathbf{Q}$)
  - when we know that the gras is wet ($\mathbf{E}$)

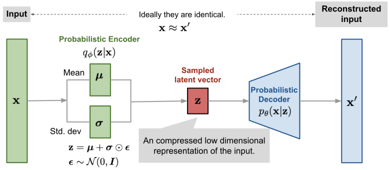
  $P(\text{Has Rained} = \text{true} \mid \text{Gras} = \text{wet})$

Probabilistic Models

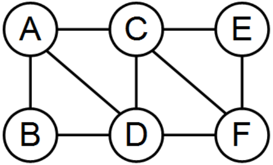
Bayesian Networks



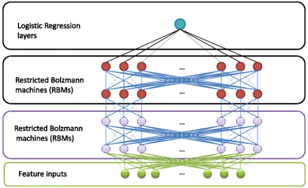
Variational Autoencoder



Markov Networks



Deep Belief Networks



Outline

Key Ideas

Probabilistic programming lets users

- specify probabilistic models *as programs*
- *compile* those models down into inference procedures

Key Ideas

Probabilistic programming lets users

- specify probabilistic models *as programs*
- *compile* those models down into inference procedures

Two compositional representations as *first class citizens*

- Random variables
- Inference

Key Ideas

Probabilistic programming lets users

- specify probabilistic models *as programs*
- *compile* those models down into inference procedures

Two compositional representations as *first class citizens*

- Random variables
- Inference

Goal

Make probabilistic programming as **flexible** and **efficient** as deep learning!

Typicall PPL Tradeoffs

Probabilistic programming languages typically have the following trade-off:

Typicall PPL Tradeoffs

Probabilistic programming languages typically have the following trade-off:

- Expressiveness
  - allow *rich class* beyond graphical models
  - *scales poorly* w.r.t. data and model size

Typicall PPL Tradeoffs

Probabilistic programming languages typically have the following trade-off:

- Expressiveness
  - allow *rich class* beyond graphical models
  - *scales poorly* w.r.t. data and model size
- Efficiency
  - PPL is restricted to a *specific class* of models
  - inference algorithms are *optimized* for this specific class

Edward

Edward (Tran et al. 2017) builds on two compositional representations

- Random variables
- Inference

Edward

Edward (Tran et al. 2017) builds on two compositional representations

- Random variables
- Inference

Edward allows to fit the same model using a variety of *composable* inference methods

- Point estimation
- Variational inference
- Markov Chain Monte Carlo

Edward

Key concept: no distinct model or inference block

- *Model*: Composition/collection of random variables
- *Inference*: Way of modifying parameters in that collection subject to another

Edward

Uses computational benefits from TensorFlow like

- distributed training
- parallelism
- vectorization
- GPU support “for free”

Outline

Criteria for Probabilistic Models

Edward poses the following criteria on compositional representations for **probabilistic models**:

1. Integration with *computational graphs*
   - nodes represent operations on data
   - edges represent data communicated between nodes

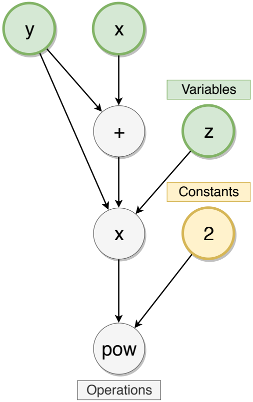
Criteria for Probabilistic Models

Edward poses the following criteria on compositional representations for **probabilistic models**:

1. Integration with *computational graphs*
   - nodes represent operations on data
   - edges represent data communicated between nodes
2. Invariance of the representation under the graph
   - graph can be reused during inference

Graph Example

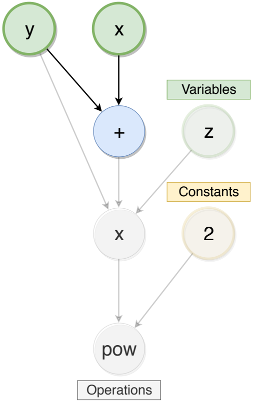
Computational Graph



Evaluation

Graph Example

Computational Graph

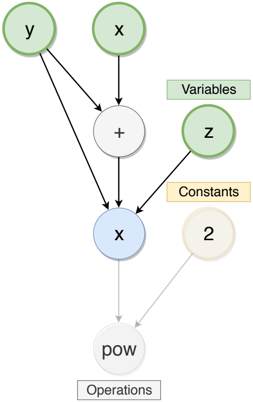


Evaluation

1. $x + y$

Graph Example

Computational Graph

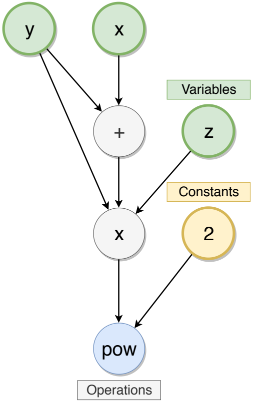


Evaluation

1. $x + y$
2. $(x + y) \cdot y \cdot z$

Graph Example

Computational Graph



Evaluation

1. $x + y$
2. $(x + y) \cdot y \cdot z$
3. $2^{(x+y) \cdot y \cdot z}$

Example: Beta-Bernoulli Programm

Beta-Bernoulli Model

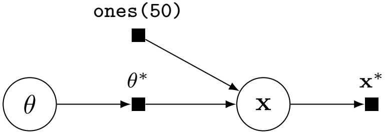
$$p(\mathbf{x}, \theta) = Beta(\theta \mid 1, 1) \prod_{n=1}^{50} Bernoulli(x_n \mid \theta)$$

Example: Beta-Bernoulli Programm

Beta-Bernoulli Model

$$p(\mathbf{x}, \theta) = Beta(\theta \mid 1, 1) \prod_{n=1}^{50} Bernoulli(x_n \mid \theta)$$

Computation Graph

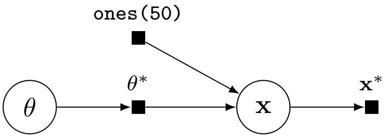


Example: Beta-Bernoulli Programm

Beta-Bernoulli Model

$$p(\mathbf{x}, \theta) = Beta(\theta \mid 1, 1) \prod_{n=1}^{50} Bernoulli(x_n \mid \theta)$$

Computation Graph



Edward code

```
theta = Beta(a=1.0, b=1.0)              # Sample from Beta dist.
x = Bernoulli(p=tf.ones(50) * theta)    # Sample from Bernoulli dist.
```

Criteria for Probabilistic Inference

Edward poses the following criteria on compositional representations for **probabilistic inference**:

1. Support for many classes of inference

Criteria for Probabilistic Inference

Edward poses the following criteria on compositional representations for **probabilistic inference**:

1. Support for many classes of inference
2. Invariance of inference under the computational graph
   - posterior can be further composed as part of another model

Inference in Edward

Goal: calculate posterior $p(\mathbf{z}, \beta \mid \mathbf{x}_{train}; \boldsymbol{\theta})$, given

- data $\mathbf{x}_{train}$
- model parameters $\boldsymbol{\theta}$
- local variables $\mathbf{z}$
- global variables β

Inference as Stochastic Graph Optimization

Edward formalize this as *optimization problem*

$$\min_{\boldsymbol{\lambda},\boldsymbol{\theta}} \mathcal{L}(p(\mathbf{z}, \beta \mid \mathbf{x}_{train}; \boldsymbol{\theta}),\ q(\mathbf{z}, \beta; \boldsymbol{\lambda}))$$

where

- $\mathcal{L}$ is a loss function w.r.t. p and q
- $q(\mathbf{z}, \beta; \boldsymbol{\lambda})$ is an approximation of the posterior $p(\mathbf{z}, \beta \mid \mathbf{x}_{train}; \boldsymbol{\theta})$

Note

Choice of approximation q, loss $\mathcal{L}$ and rules to update parameters $\{\boldsymbol{\theta}, \boldsymbol{\lambda}\}$ are specified by an inference algorithm.

Inference in Edward

- `ed.Inference` defines and solves $\min_{\boldsymbol{\lambda},\boldsymbol{\theta}} \mathcal{L}(p(\mathbf{z}, \beta \mid \mathbf{x}_{train}; \boldsymbol{\theta}),\ q(\mathbf{z}, \beta; \boldsymbol{\lambda}))$

```
# Construct inference object
inference = ed.Inference(latent_vars={beta: qbeta, z:qz},
                         data={x: x_train})
```

  - Posterior variables: `qbeta`, `qz`, Observed random variables: `x_train`

Inference in Edward

- `ed.Inference` defines and solves $\min_{\boldsymbol{\lambda},\boldsymbol{\theta}} \mathcal{L}(p(\mathbf{z}, \beta \mid \mathbf{x}_{train}; \boldsymbol{\theta}),\ q(\mathbf{z}, \beta; \boldsymbol{\lambda}))$

```
# Construct inference object
inference = ed.Inference(latent_vars={beta: qbeta, z:qz},
                         data={x: x_train})
```

  – Posterior variables: `qbeta`, `qz`, Observed random variables: `x_train`

- Build a computational graph to update parameters

```
inference.initialize()
```

Inference in Edward

- `ed.Inference` defines and solves $\min_{\boldsymbol{\lambda},\boldsymbol{\theta}} \mathcal{L}(p(\mathbf{z},\beta \mid \mathbf{x}_{train};\boldsymbol{\theta}),\ q(\mathbf{z},\beta;\boldsymbol{\lambda}))$

```
# Construct inference object
inference = ed.Inference(latent_vars={beta: qbeta, z:qz},
                         data={x: x_train})
```

  - Posterior variables: `qbeta`, `qz`, Observed random variables: `x_train`

- Build a computational graph to update parameters

```
inference.initialize()
```

- Run computations to update parameters

```
while not_converged:
    inference.update()
```

Classes of Inference

Edward supports the following classes of inference:

- Variational Inference
- Monte Carlo
- Generative Adverserial Networks (GANs)

Composing Inferences

Inference as a collection of separate inference programs, e.g. **Variational EM:**

```
qbeta = PointMass(...)   # Global variables
qz    = Categorical(...) # Local variables
```

Composing Inferences

Inference as a collection of separate inference programs, e.g. **Variational EM:**

```
qbeta = PointMass(...)   # Global variables
qz   = Categorical(...)  # Local variables

# E-Step over local variables
inf_e = ed.VariationalInference(latent_vars={z: qz},
                                    data={x: x_train, beta: qbeta})
# M-Step over global variables
inf_m = ed.MAP(latent_vars={beta: qbeta},
                     data={x: x_train, z: qz})
```

Composing Inferences

Inference as a collection of separate inference programs, e.g. **Variational EM:**

```
qbeta = PointMass(...)  # Global variables
qz   = Categorical(...)  # Local variables

# E-Step over local variables
inf_e = ed.VariationalInference(latent_vars={z: qz},
                                data={x: x_train, beta: qbeta})
# M-Step over global variables
inf_m = ed.MAP(latent_vars={beta: qbeta},
               data={x: x_train, z: qz})

# Expectation-Maximization loop
while not_converged:
  inf_e.update()
  inf_m.update()
```

Outline

Benchmarks

Logistic Regression using Hamiltonian Monte Carlo iterations

Probabilistic programming system	Runtime (s)
Handwritten NumPy (1 CPU)	534
Stan (1 CPU) (Carpenter et al. 2017)	171
PyMC3 (12 CPU) (Salvatier et al. 2015)	30.0
Edward (12 CPU)	**8.2**
Handwritten TensorFlow (GPU)	5.0
Edward (GPU)	**4.9**

- 35x Speedup over Stan (1 CPU)
- 6x Speedup over PyMC3 (12 CPU)

(CPU: 12-core Intel i7-5930K at 3.50GHz, GPU: NVIDIA Titan X (Maxwell))

Outline

Edward Successor: TensorFlow Probability (Dillon et al. 2017)

Integration into TensorFlow itself: 4-Layer architecture

1. **TensorFlow** – Numerical operations
2. **Statistical Building Blocks** – Distributions
3. **Model Building** – Joint distributions, Probabilistic layers
4. **Probabilistic Inference** – Markov Chain Monte Carlo, Variational inference, Optimizers

Pyro: PyTorch Probabilistic Programming (Bingham et al. 2018)

PYRO

- *PyTorch* as backend
- Unifies modern deep learning and Bayesian modeling
- Focus on Stochastic Variational Inference

Outline

Conclusion

Edward . . .

- is a novel deep probabilistic programming language
- provides **compositional representations** for model and inference
- leverages **computational graphs** for fast parallelizable computation

References I

- Bingham, Eli et al. (2018). "Pyro: Deep Universal Probabilistic Programming". In: *Journal of Machine Learning Research*.
- Carpenter, Bob et al. (2017). "Stan: A Probabilistic Programming Language". In: *Journal of Statistical Software, Articles* 76.1, pp. 1–32. ISSN: 1548-7660. DOI: 10.18637/jss.v076.i01. URL: https://www.jstatsoft.org/v076/i01.
- Dillon, Joshua V. et al. (2017). *TensorFlow Distributions*. arXiv: 1711.10604 [cs.LG].
- He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385. arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.
- Lecun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*, pp. 2278–2324.
- Salvatier, John et al. (2015). *Probabilistic Programming in Python using PyMC*. arXiv: 1507.08050 [stat.CO].

References II

- Simonyan, Karen and Andrew Zisserman (2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556. URL: http://arxiv.org/abs/1409.1556.
- Szegedy, Christian et al. (2014). "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842. arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.
- Tran, Dustin et al. (2017). *Deep Probabilistic Programming*. arXiv: 1701.03757 [stat.ML].

Figure Sources

- **CNNs**: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d
- **Bayesian Networks**: K. Kersting, Probabilistic Graphical Models Lecture (2.), 2018
- **Markov Models**: https://en.wikipedia.org/wiki/File:A_simple_Markov_network.png
- **Variational Autoencoder**: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html
- **Deep Belief Networks**: https://medium.com/analytics-army/deep-belief-networks-an-introduction-1d52bb867a25

Example: Variational Auto-Encoder

```
# Probabilistic model
z = Normal(loc=tf.zeros([50, 10]), scale=tf.ones([N, 10]))
h = Dense(256, activation="relu")(z)
x = Bernoulli(logits=Dense(28 * 28)(h))

# Variational model
qx = tf.placeholder(tf.float32, [50, 28 * 28])
qh = Dense(256, activation="relu")(qx)
qz = Normal(loc=Dense(10, activation=None) (qh),
            scale=Dense(10, activation="softplus")(qh))
```