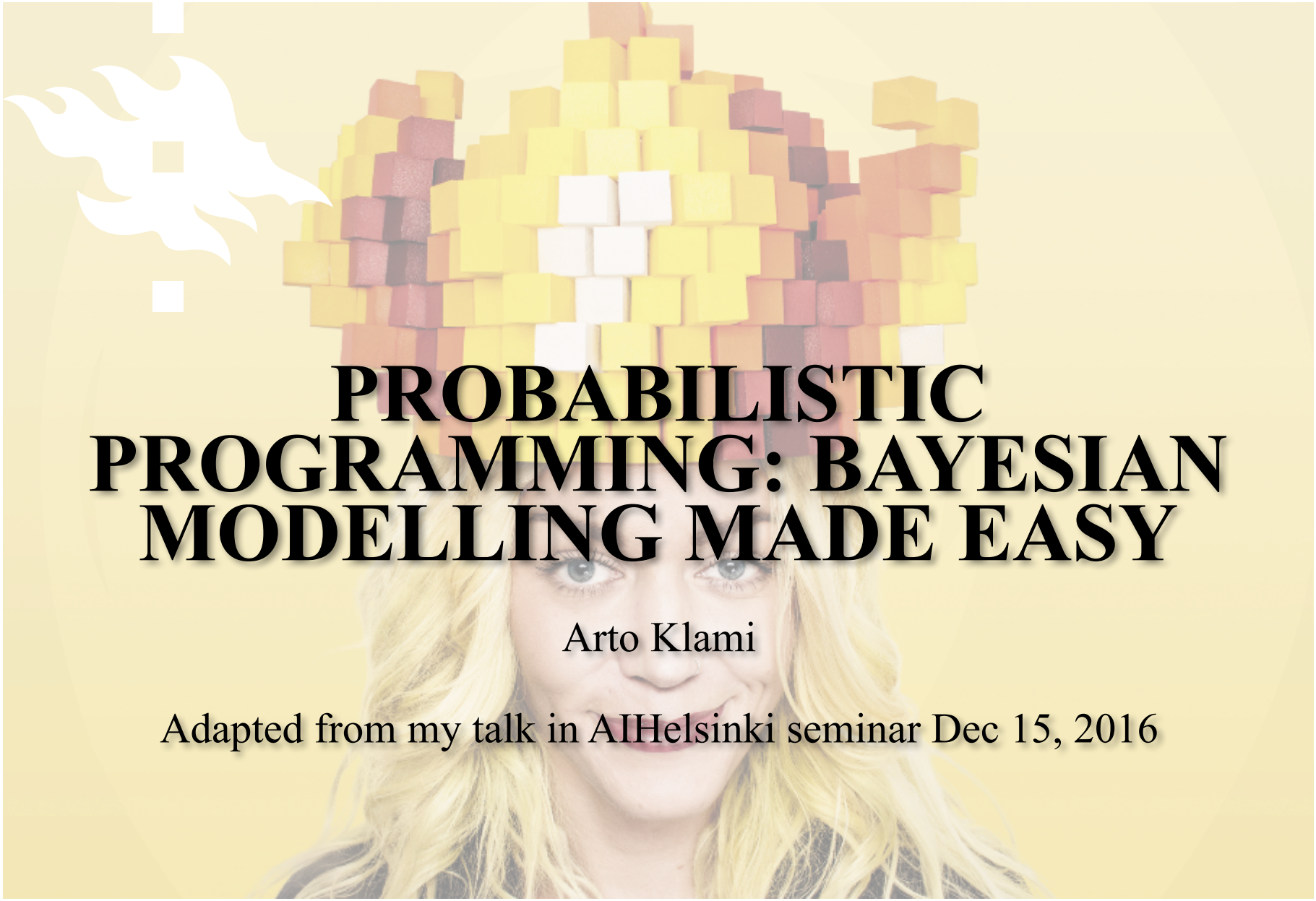


PROBABILISTIC PROGRAMMING: BAYESIAN MODELLING MADE EASY

Arto Klami

Adapted from my talk in AIHelsinki seminar Dec 15, 2016

MOTIVATING INTRODUCTION

Most of the artificial intelligence success stories today use deep learning and neural networks

…that were largely invented decades ago:

- Multi-layer perceptron 60’s - 80’s
- Modern convolutional neural networks around 1998
- Modern recurrent neural networks (LSTM) around 1997

WHY NOW?

1. More data
2. More computing power (especially GPU)
3. Small but crucial algorithmic improvements
4. Better tools
   - Create complex models out of simple building blocks
   - Learning largely automatic, robust and verified software

DEEP LEARNING

```
# Specify the model
result, loss = (pretty_tensor.wrap(input_data, m).flatten()
      .fully_connected(200, activation_fn=tf.nn.relu)
      .fully_connected(10, activation_fn=None)
      .softmax(labels, name=softmax_name))


# Pick optimizer
optimizer = tf.train.GradientDescentOptimizer(0.1)


# Learn the model
train_op = pt.apply_optimizer(optimizer, losses=[loss])
```

PROBABILISTIC PROGRAMMING

"*Probabilistic programming is to probabilistic modelling as deep learning is to neural networks*" (Antti Honkela, 2016)

- Delivers the compositionality probabilistic graphical models always promised
- Provides the tools we need to speed up development and to support more flexible models

PROBABILISTIC MODELS

A probabilistic model tells a generative story and is defined via collection of probability distributions

Examples:

- Mixture models
- Hidden Markov models
- Markov random fields
- Logistic regression, generalized linear models
- Most neural networks (MLP, CNN, etc.)

PROBABILISTIC MODELS

Latent Dirichlet allocation (LDA) is a generative model for text

Choose topics

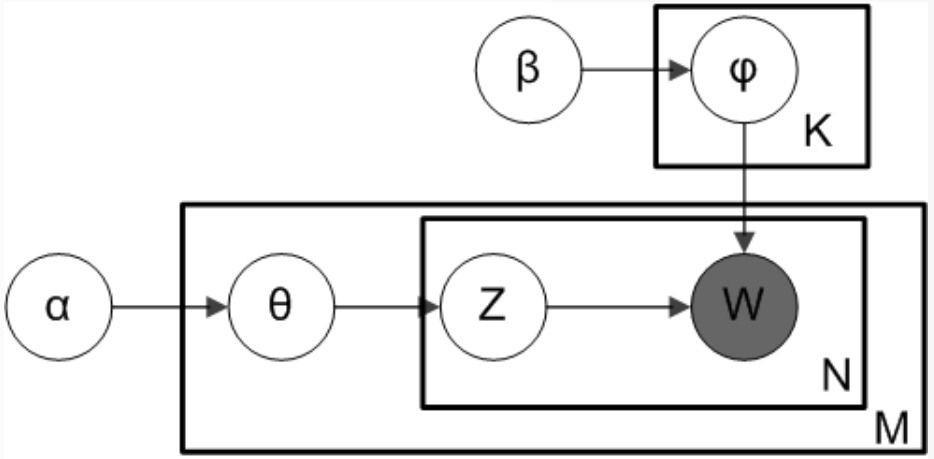
$$\psi_k \sim \mathrm{Dir}(\beta)$$

For each document

- Choose topic proportions

$$\theta_d \sim \mathrm{Dir}(\alpha)$$

- Loop over the document
  - Choose topic $z_{d,i} \sim \mathrm{Cat}(\theta_d)$
  - Choose word $w_{d,i} \sim \mathrm{Cat}(\psi_{z_{d,i}})$



INFERENCE

The prior $p(\theta)$ encodes our uncertainty before seeing data

The likelihood $p(x|\theta)$ tells how likely the data is given the parameters

The Bayes rule $p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} = \frac{p(x|\theta)p(\theta)}{\int p(x|\theta)p(\theta)d\theta}$

gives the posterior that captures the uncertainty after seeing the data

Predictions by averaging over the posterior:

$$p(\hat{x}) = \int p(\hat{x}|\theta)p(\theta|x)d\theta$$

WHY BOTHER?

Can't we just learn the best θ by maximizing the likelihood?

With enough data we can, especially when we only care about (mean) predictions

This is what most deep learning solutions do, maximize likelihood (regularized by some priors) using SGD

WHY BOTHER?

Parameter inference:

- We want to understand parameters of some process or system
- Natural science, economy, social science, etc.

Quantification of uncertainty:

- Limited data, most of the time also in “big data” cases
- Decision-making, avoiding false alarms

Often unsupervised, but need not be

PARAMETER INFERENCE

"Should I eat more now than in the past?"

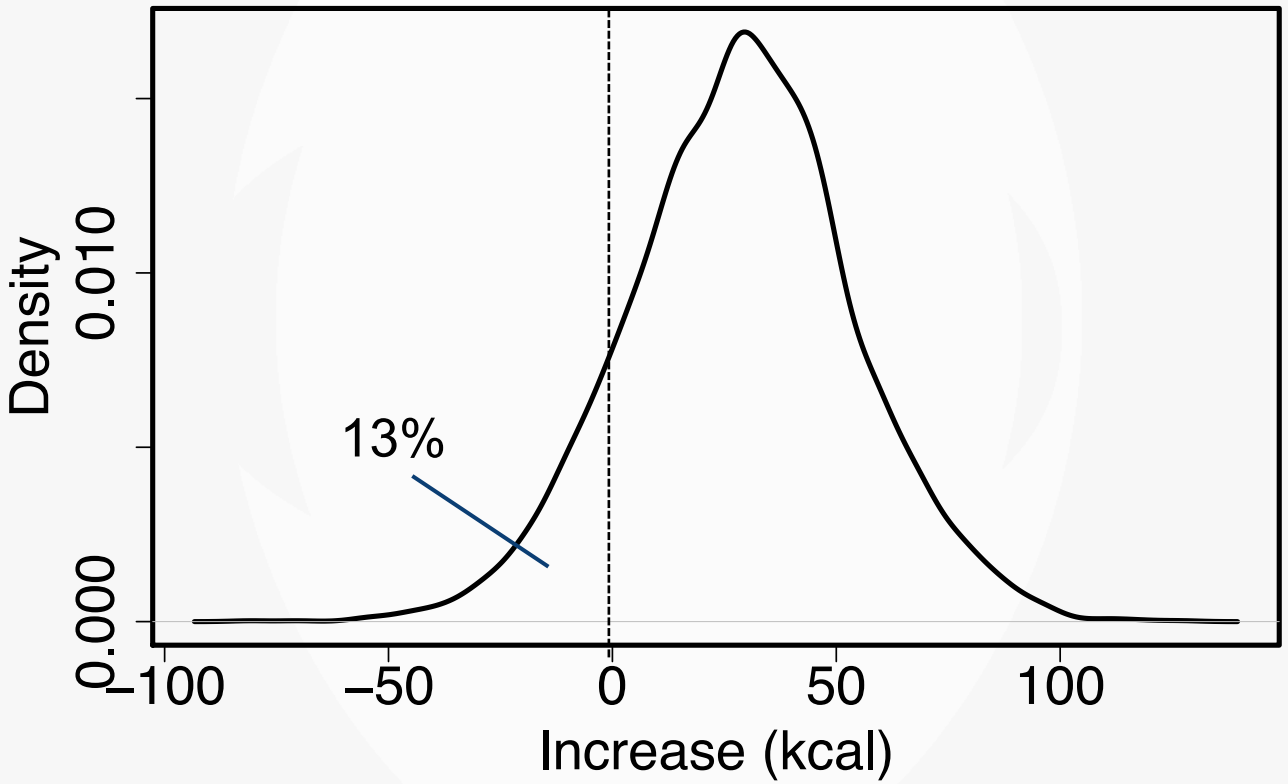
My phone/watch records energy consumption, so let's compare the change between two consecutive months:

- September: Mean 1912kcal (above basic consumption)
- October: Mean 1946kcal

Did it actually raise?

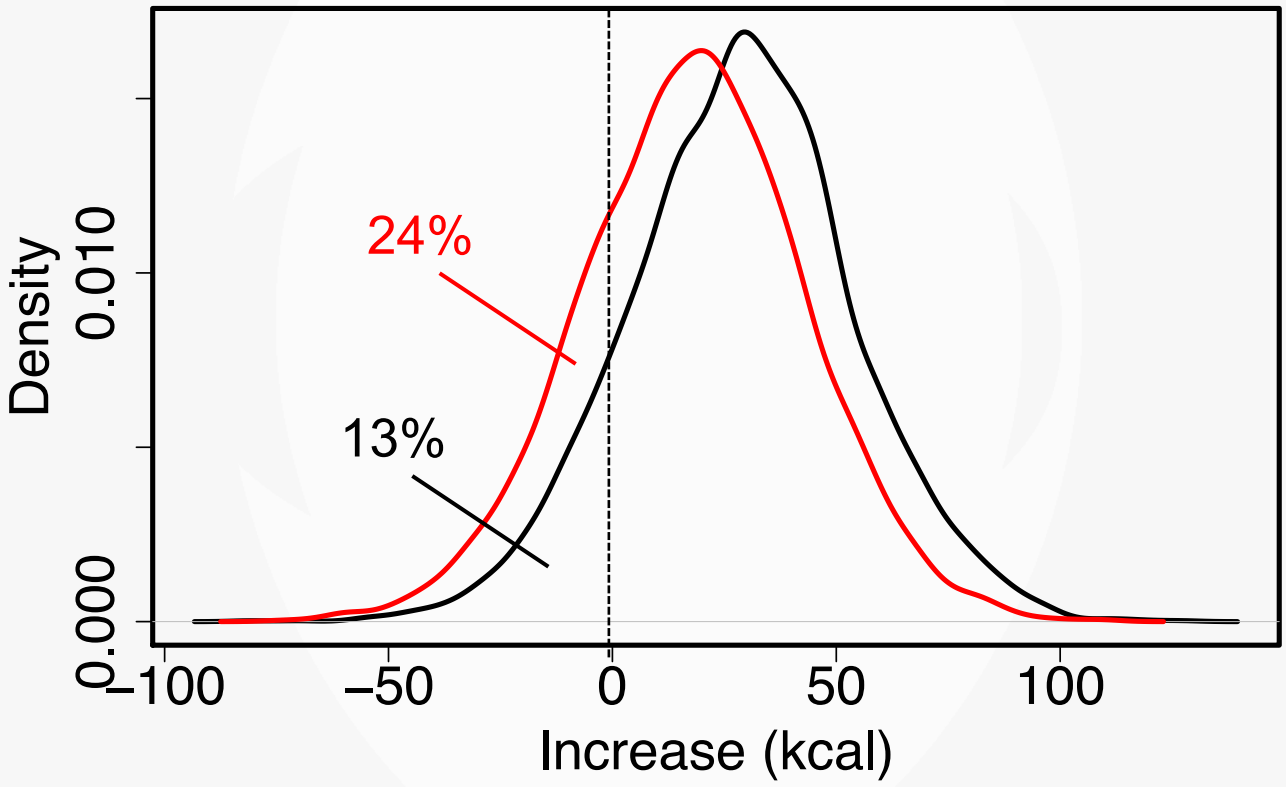
- Write down a model for the daily observations and infer the mean rate

PARAMETER INFERENCE



$$x \sim N(\mu, \sigma)$$

PARAMETER INFERENCE

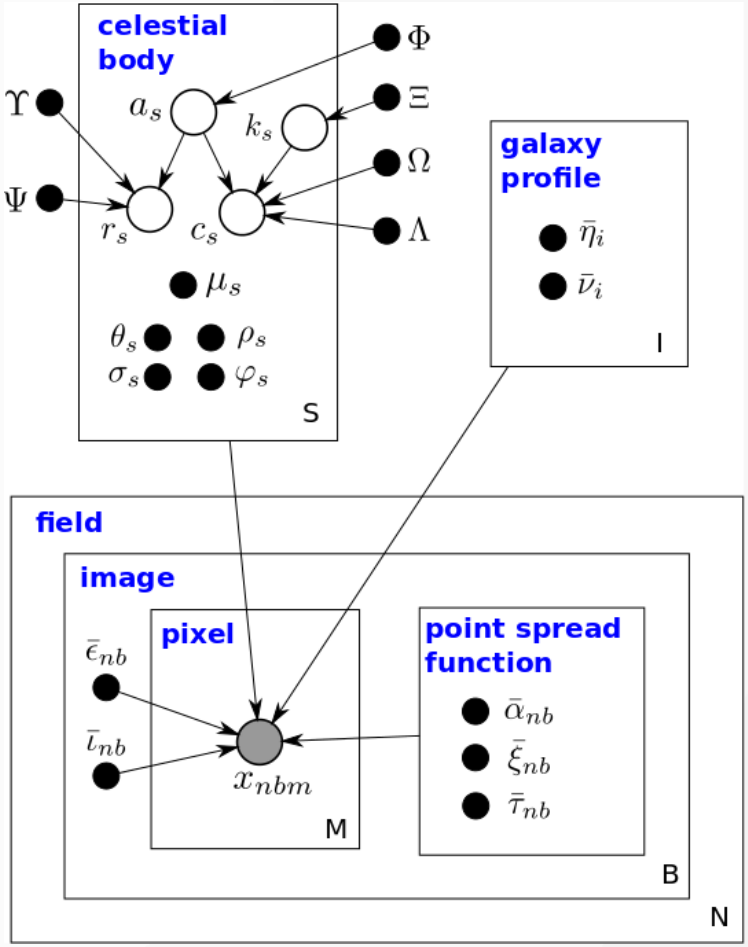


$x \sim N(\mu, \sigma)$

$x \sim t_\nu(\mu, \sigma)$

COUNTING GALAXIES
[REGIER ET AL. ICML'15]

Figure 1. An image from the Sloan Digital Sky Survey (SDSS, 2015) of a galaxy from the constellation Serpens, 100 million light years from Earth, along with several other galaxies and many stars from our own galaxy.



WHAT'S THE CHALLENGE?

The Bayes rule looks elegant, but computing the evidence

$$p(x) = \int p(x, \theta) d\theta$$

is ridiculously hard for all interesting models

Markov chain Monte Carlo (MCMC), variational approximation, expectation propagation etc.

The ML community has spent decades developing specific inference algorithms applicable for individual (more and more complex) model families

WHAT'S THE CHALLENGE?

according to the above probability, we do not need the exact value of

$$P\left(Z_{m,n} \mid \boldsymbol{Z}_{-(\boldsymbol{m},\boldsymbol{n})}, \boldsymbol{W}; \alpha, \beta\right)$$

but the ratios among the probabilities that $Z_{(m,n)}$ can take value. So, the above equation can be simplified as:

$$P(Z_{(m,n)} = k \mid \boldsymbol{Z}_{-(m,n)}, \boldsymbol{W}; \alpha, \beta)$$

$$\propto P(Z_{(m,n)} = k, \boldsymbol{Z}$$

$$= \left(\frac{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \right) \prod_{i=1}^{K} \frac{\Gamma\left(n^i_{(\cdot),v} + \beta_v\right)}{\Gamma\left(\sum_{r=1}^{V} n^i_{(\cdot),r} + \beta_r\right)}$$

$$\propto \frac{\prod_{i=1}^{K} \Gamma\left(n^i_{m,(\cdot)}\right.}{\Gamma\left(\sum_{i=1}^{K} n^i_{m,(\cdot)}\right.}$$

$$\propto \prod_{i=1}^{K} \Gamma\left(n^i_{m,(\cdot)} + \alpha_i\right) \prod_{i=1}^{K} \frac{}{\Gamma\left(\sum_{r=1}^{V} n^i_{(\cdot),r} + \beta_r\right)}.$$

Even slight modifications of the model require derivations worth of a research paper, often way more complex than the original algorithm.

Finally, let $n^{i,-(m,n)}_{j,r}$ be the same meaning as $n^i_{j,r}$ but with the $Z_{(m,n)}$ excluded. The above equation can be further simplified leveraging the property of gamma function. We first split the summation and then merge it back to obtain a k-independent summation, which could be dropped:

MODELS AND INFERENCE

We should separate the model and inference, letting people focus on writing interesting models without needing to worry (too much) about inference

Being able to do this was big part of the deep learning resurgence, and probabilistic models are naturally suited for modularity

Just a bit of a delay, because the problem is harder:

- I started my ML career in 2000, deriving backprob and implementing a MLP in Matlab; that would now be 5 lines of code
- Around 2010 I was deriving variational approximations and implementing them in R; that should now be 10 lines of code

PP TO THE RESCUE

```
parameters {
  simplex[K] theta[M];   // topic dist for doc m
  simplex[V] psi[K];     // word dist for topic k
}
model {
  for (m in 1:M)
    theta[m] ~ dirichlet(alpha);  // prior
  for (k in 1:K)
    psi[k] ~ dirichlet(beta);     // prior
  for (n in 1:N) {
    real gamma[K];
    for (k in 1:K)
      gamma[k] <- log(theta[doc[n],k]) + log(psi[k,w[n]]);
    increment_log_prob(log_sum_exp(gamma));  // likelihood
  }
}
```

LDA in Stan [Carpenter et al. JASS'16]

WHAT'S UNDER THE HOOD?

MCMC still the basic tool, but we now know how to implement gradient-based MCMC algorithms (Hamiltonian MC) efficiently

- Gradients computed by automatic differentiation of the log probability, proposal lengths determined automatically etc.

Variational approximation also a hot topic right now

- Maximizes a lower bound for the evidence
- Gradient-based optimization possible via reparameterization

Both problems still harder than ML estimation

WHAT DID WE GAIN?

Inference happens quite automatically

…and we gained a lot of flexibility:

1. Non-conjugate models became just as easy as conjugate ones
2. Arbitrary control flows (while-loops etc) as part of the model
3. (Quite) arbitrary differentiable random procedures as part of the generative story (e.g. rendering an image)

Want to try today? Download Stan and use HMC for inference

WHAT'S STILL MISSING?

ML community has produced plenty of dedicated tools for structured models that are much more efficient than black-box optimization

- E.g., dynamic programming for inferring the latent states of a HMM
- We need to better integrate those as part of the inference process for arbitrary models

Scalability and accuracy

- HMC is good, but still slow
- Variational approximations still not sufficiently accurate for all models

PP VS DEEP LEARNING

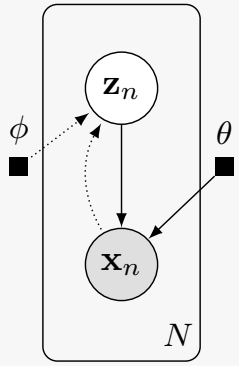
Deep learning models are probabilistic models

Bayesian inference is harder, but many of the things we have learned about gradient-based optimization still apply

- We can implement PP tools using the same platforms: Edward uses TensorFlow, PyMC3 uses Theano

My prediction: In 5 years or so, the tools will be the same, and the choice of the model and inference algorithm depends on the application and available data

PP VS DEEP LEARNING




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

# Variational model
qx = tf.placeholder(tf.float32, [N, 28 * 28])
qh = slim.fully_connected(qx, 256)
qz = Normal(mu=slim.fully_connected(qh, d, activation_fn=None),
            sigma=slim.fully_connected(qh, d, activation_fn=tf.nn.softplus))
```

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.

```
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})
```

```
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})
```

Figure 4: (left) Variational inference. (right) Monte Carlo.

VAE in Edward [Tran et al. arXiv'16]

PP IN HELSINKI

My group is working on scalable variational approximations

- *Scalable probabilistic analytics* (Tekes, Reaktor, M-Brain, Ekahau)

Aki Vehtari (Aalto) is part of Stan core development team, focusing in particular on model assessment

Aalto+UH: ELFI for approximative Bayesian computation (ABC, kind of PP but uses simulators as models)