An intro* to probabilistic programming (using Pyro)

*For people who know deep learning

MLRG Talk, June 11, 2020

Shashank Shekhar

Credits

Parts of slides/ideas adapted from:

- Maria Gorinova, University of Edinburgh, Probabilistic Programming: The What, Why and How
- Shakir Mohamed, DeepMind, Probabilistic Reasoning & Variational Inference: Foundations | Tricks | Algorithms

Code adapted/borrowed from: Pyro documentation

Thanks to Mike for reviewing the tutorial

Pre-requisites

- Introductory probability
- Introductory machine learning
- Python programming
- Some familiarity with Pytorch

In this tutorial...

- Probabilistic Programming: Breaking down the two components
- Bayesian Inference: Reframing the architecture-loss f/w of learning into the model-inference-algorithm f/w
- ~~Sampling based inference~~*
- Variational Inference
- Deep learning case study: VAEs (time permitting)

*I will add notebooks and post the complete slides with sampling based methods on Slack. This tutorial will mention them only in passing (for completeness) in the interest of time.

Disclaimer (long)

Author is an electrical engineer by training and (to his chagrin) not a:

- **Computer scientist:** Never took a class on programming languages, compiler design, automata theory, logic etc. (might not completely understand or explain the ideas behind the actual implementation of these languages)
- **Statistician:** Took one class on statistics and attended one summer school on bayesian machine learning (might not completely understand or explain the ideas behind all the inference mechanisms used in these languages)

(Feel free to jump in if you find something fishy. I've also referred to resources for further viewing/reading at the end)

Before we start

Please find the notebooks used in this tutorial as well as the Google Colab notebook links here:

https://github.com/sshkhr/ppl_tutorial

Probabilistic / Programming

Probabilistic Programming

Daniel M. Roy

Department of Statistical Sciences
Department of Computer Science
University of Toronto

Workshop on Uncertainty in Computation
2016 Program on *Logical Structures in Computation*
Simons Institute for the Theory of Computing

1. Simple story:
   Probabilistic programming automates Bayesian inference
2. Real story:
   It's complicated

1/50

Let's hear from the experts*....

*PhD dissertation: "Computability, inference and modeling in probabilistic programming"

Probabilistic Programming

Daniel M. Roy

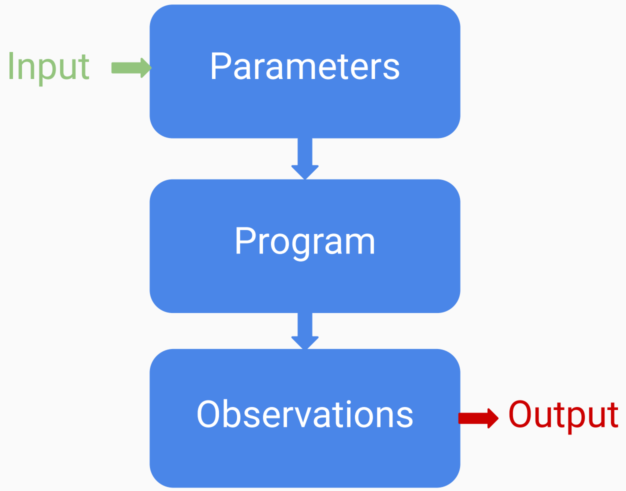
Department of Statistical Sciences
Department of Computer Science
University of Toronto

Workshop on Uncertainty in Computation
2016 Program on *Logical Structures in Computation*
Simons Institute for the Theory of Computing

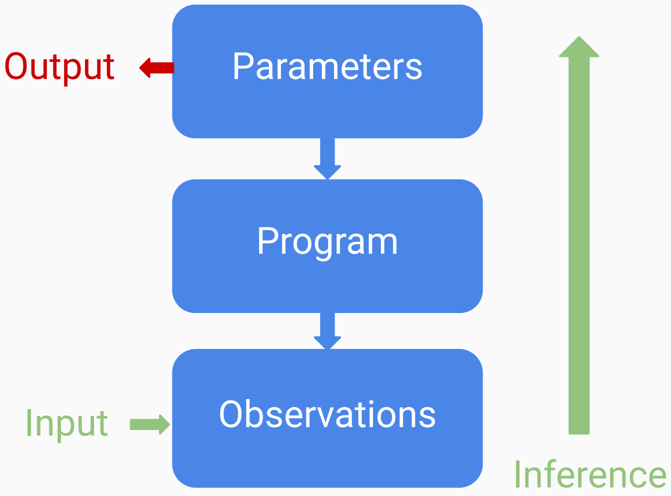
1. Simple story:
   Probabilistic programming automates Bayesian inference
2. Real story:
   It's complicated

This presentation/tutorial only covers part 1

A program



A probabilistic program



Probabilistic programming

“The key insight in PP is that statistical modeling can, when you do it enough, start to feel a lot like programming. If we make the leap and actually use a real language for our modeling, many new tools become feasible. We can start to automate the tasks that used to justify writing a paper for each instance.”

Why probabilistic programming?

- Quantify uncertainty
- Encode structure about the world through Bayesian modelling (later)
- Works well in low-data regime
- Separate modelling from inference <-- Bayesian Inference ++
- Utilize programming structures like control flow, modularity etc <-- Bayesian Inference ++

Deep learning libraries: High-level interface to actual implementation of the architectures at low-level linear algebra operations or the learning mechanism via gradient propagation

Probabilistic programming: High-level interface to modelling and inference in a Bayesian setting

Probabilistic programming vs Deep Learning

$F(x) = y$

Conventional Programming	Deep Learning (Differentiable Programming?)	Probabilistic Programming
• Given: Input x, Deterministic/ pseudo-random function F • Want: Output y	• Given: Input x, Output y • Want: (Deterministic/ random) function F	• Given: Output y, Random function F • Want: Probability distribution on input x

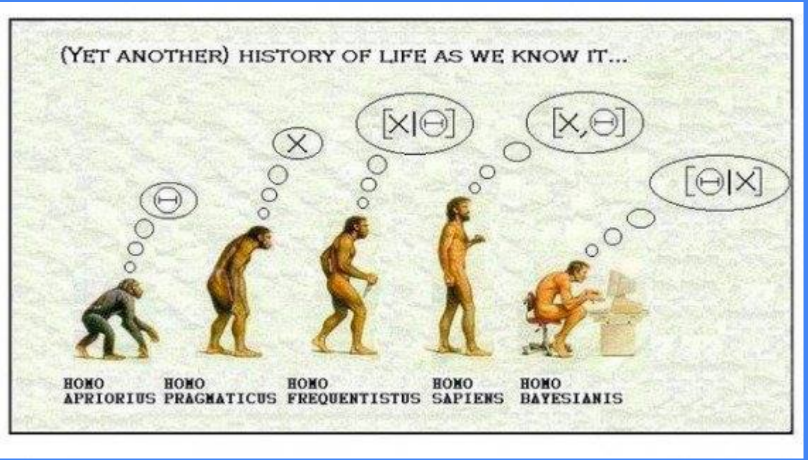
A hands-on intro to Pyro

Please follow this Google Colab link

https://colab.research.google.com/drive/1MdNRUhWtRPDjB_2M0cPEzOL1AUbEFVke?usp=sharing

File -> Save a copy in Drive = create your own copy to work with

Think Bayesian



Bayes Rule

Likelihood

Prior

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Posterior

Evidence

Bayesian inference

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)}$$

Given i.i.d data X = (X1, …, Xn) from distribution p(X|Θ) encode uncertainty about Θ in a prior p(Θ) and apply Bayesian inference:

$$p(\theta|X) = \frac{\prod_{i=1}^{n} p(x_i|\theta)\, p(\theta)}{\int \prod_{i=1}^{n} p(x_i|\theta)\, p(\theta) d\theta}$$

Full Bayesian inference

Training stage:

$$p\left(\theta|X_{tr}, Y_{tr}\right) = \frac{p\left(Y_{tr}|X_{tr}, \theta\right) p(\theta)}{\int p\left(Y_{tr}|X_{tr}, \theta\right) p(\theta) d\theta}$$

Testing stage:

$$p\left(y|x, X_{tr}, Y_{tr}\right) = \int p(y|x, \theta) p\left(\theta|X_{tr}, Y_{tr}\right) d\theta$$

Full Bayesian inference <- HARD

Training stage:

$$p\left(\theta | X_{tr}, Y_{tr}\right) = \frac{p\left(Y_{tr} | X_{tr}, \theta\right) p(\theta)}{\int p\left(Y_{tr} | X_{tr}, \theta\right) p(\theta) d\theta}$$

These integrals can be intractable

Testing stage:

$$p\left(y | x, X_{tr}, Y_{tr}\right) = \int p(y | x, \theta) p\left(\theta | X_{tr}, Y_{tr}\right) d\theta$$

Full Bayesian inference <- HARD

Training stage:

$$p\left(\theta|X_{tr},Y_{tr}\right)=\frac{p\left(Y_{tr}|X_{tr},\theta\right)p(\theta)}{\int p\left(Y_{tr}|X_{tr},\theta\right)p(\theta)d\theta}$$

These integrals can be intractable

We shall see how to deal with them in the next sections

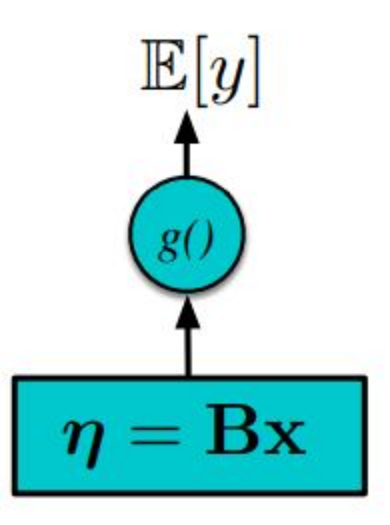
Testing stage:

$$p\left(y|x,X_{tr},Y_{tr}\right)=\int p(y|x,\theta)p\left(\theta|X_{tr},Y_{tr}\right)d\theta$$

Moving from DL to Bayesian Inference

In deep learning we think of our model as the network architecture. We frame a loss function which on minimization gives a point estimate (usually maximum likelihood or penalized maximum likelihood) of the model parameters (network weights).

Architecture - Loss f/w: Linear Regression

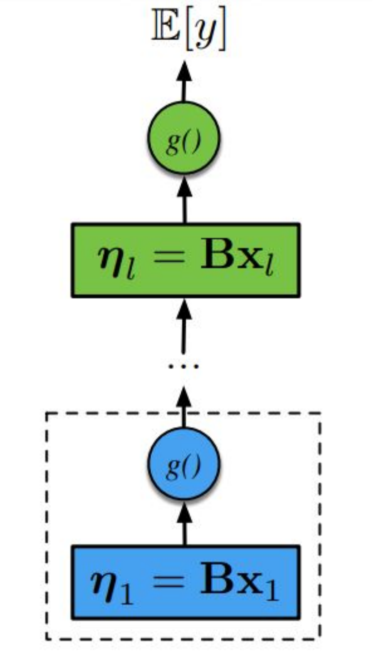


Architecture: $y = w^T x + b$

Loss: Least squares loss (+ regularization)

Optimization: Normal equations (analytical) / Gradient descent

Architecture - Loss f/w: Deep Learning



Architecture: $E[y] = h_L * h_{L-1} \ldots\ldots * h_l * h_0(x)$

Loss: Least squares loss (+ regularization)

Optimization: Stochastic gradient descent

Least squares regression = MLE

Linear regression model ->

$$Y = X\beta + \epsilon, \text{ where } \epsilon \sim N\left(0, I\sigma^2\right)$$
$$Y \in \mathbb{R}^n, X \in \mathbb{R}^{n\times p} \text{ and } \beta \in \mathbb{R}^p$$

Likelihood for i.i.d data ->

$$L(Y|X,\beta) = \prod_{i=1}^{n} f\left(y_i|x_i,\beta\right)$$

$$L(Y|X,\beta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\vartheta_i - x_i\beta)^2}{2\sigma^2}}$$

Least squares regression = MLE (contd)

Log likelihood -> $$\sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{(y_i - x_i\beta)^2}{2\sigma^2}$$

MLE estimate -> $$\widehat{\beta}_{MLE} = \underset{\beta}{\operatorname{argmax}} \sum_{i=1}^{n} -\frac{(y_i - x_i\beta)^2}{2\sigma^2}$$

$$\widehat{\beta}_{MLE} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - x_i\beta)^2 = \widehat{\beta}_{LS}$$ <- Least squares estimate

MLE $\subseteq$ Inference

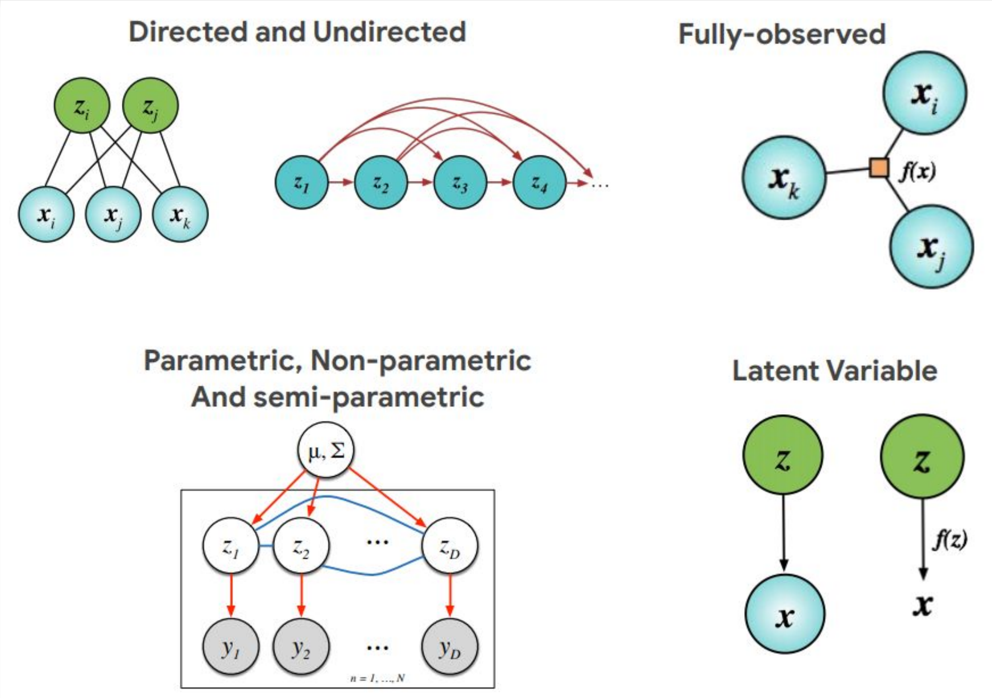
MLE = One of multiple ways to make statistical inferences

Probabilistic inference mechanisms:
- Provide a natural idea of uncertainty (sample far from the mean)
- Enable generative modelling

Bayesian inference mechanisms: provide knowledge about structure of data to model in the form of priors

Models

CNN =
Directed &
Parametric

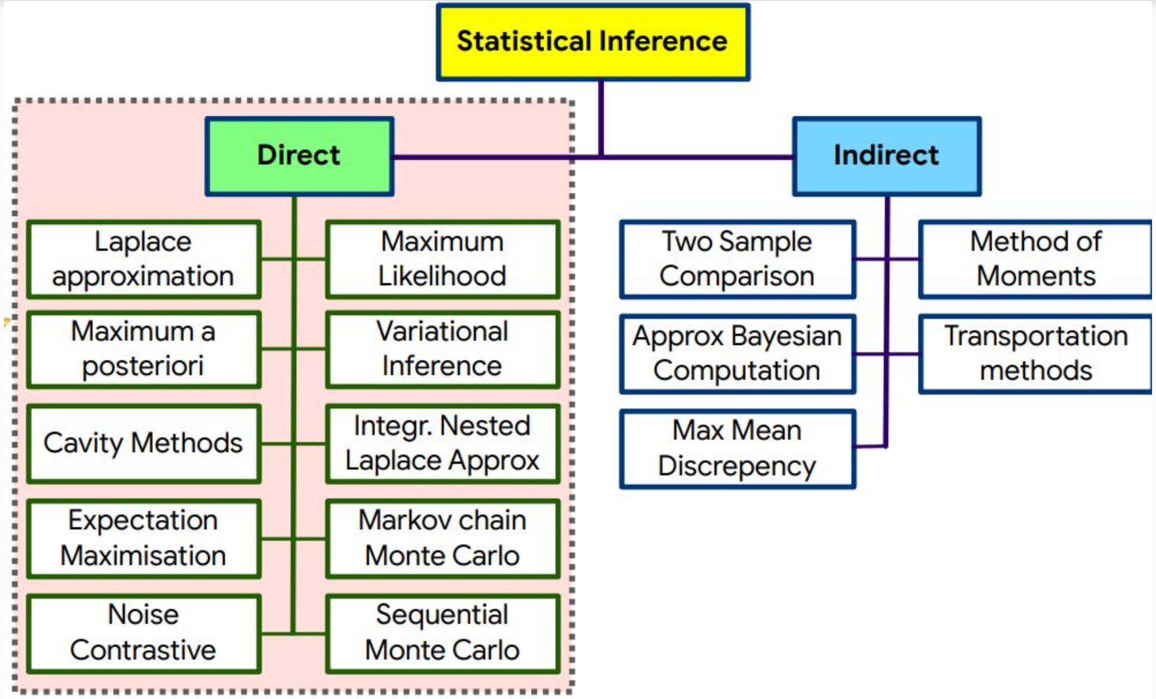


Bayes Net=
Fully-observed

Gaussian Process =
Non-parametric

VAE and GAN =
Latent Variable models

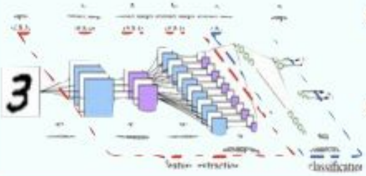
Inference



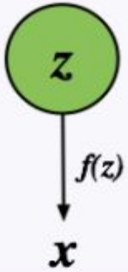
Statistical Inference
Direct
Indirect
Laplace approximation
Maximum Likelihood
Maximum a posteriori
Variational Inference
Cavity Methods
Integr. Nested Laplace Approx
Expectation Maximisation
Markov chain Monte Carlo
Noise Contrastive
Sequential Monte Carlo
Two Sample Comparison
Method of Moments
Approx Bayesian Computation
Transportation methods
Max Mean Discrepency

Model + Inference + Algorithm

Convolutional neural network + penalised maximum likelihood

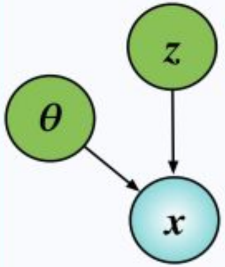


- Optimisation methods (SGD, Adagrad)
- Regularisation (L1, L2, batchnorm, dropout)



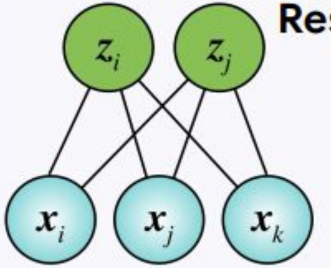
Implicit Generative Model + Two-sample testing

- Unsupervised-as-supervised learning
- Approximate Bayesian Computation (ABC)
- Generative adversarial network (GAN)



Latent variable model + variational inference

- VEM algorithm
- Expectation propagation
- Approximate message passing
- Variational auto-encoders (VAE)



Restricted Boltzmann Machine + maximum likelihood

- Contrastive Divergence
- Persistent CD
- Parallel Tempering
- Natural gradients

Bayesian Inference using Pyro

Please follow this Google Colab link

https://colab.research.google.com/drive/1m690LL-xpS1i9CNlYPY6y7SJO00SLv0J?usp=sharing

File -> Save a copy in Drive = create your own copy to work with

Sampling based Inference

Bayesian Inference = Integration

Bayesian Inference = Integration <- Hard

Bayesian Inference = Integration <- Hard

(Approximate by)

Summation <- Easy

Importance Sampling

Integral problem -> $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$

Introduce new distribution -> $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})\frac{q(\mathbf{z})}{q(\mathbf{z})}d\mathbf{z}$

Re-weight/re-group -> $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}q(\mathbf{z})d\mathbf{z}$

Transformed Integral (after importance sampling) -> $p(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z})}\left[p(\mathbf{x}|\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}\right]$

MCMC

Without going into much details:

Monte-Carlo method replaces an integral $\mathrm{E}_\pi[g(X)] = \int g(x)\pi(x)dx$ with a summation $\bar{g}_n = \frac{\sum_{i=1}^{n} g(x_i)}{n}$

A **Markov Chain** is a sequence of random variables $\{X_i\}$ where the random variable at next step X_{t+1} depends only on the last random variable X_t (first-order Markov chain) or a few of the last r.v.

In Bayesian Inference the random variables we are interested in are the model parameters.

The Ergodic theorem says that for and irreducible, aperiodic, positive recurrent Markov chain with stationary distribution, π(x): $f_n = \frac{1}{n}\sum_{t=1}^{n} f\left(x^t\right) \longrightarrow \mathbb{E}_\pi(f(x)), as\ n \rightarrow \infty$

MCMC (contd)

So putting these two together:

- We can use Monte Carlo estimates to replace an integral with a summation
- We can use Markov chains to obtain estimate of any function of a r.v. X and since many integrals we are interested in are expectations they can obtained using a Markov chain.

Thus, MCMC can be used to sample for random variables and then these samples can be summed to approximate expectations over distributions

Common MCMC methods: Metropolis Hastings, Random Walk Metropolis Hastings, Gibbs Sampling, Hamiltonian Monte Carlo, No U-Turn Sampling

Variational Inference

Bayesian Inference = Integration

Bayesian Inference = Integration <- Hard

Bayesian Inference = Integration <- Hard

(Approximate/replace by) ↑

Optimization <- Easy(ish)

VI vs MCMC

Probabilistic model $p(x, \theta) = p(x|\theta)p(\theta)$

Variational Inference

Approximate $p(\theta|x) \approx q(\theta) \in \mathcal{Q}$

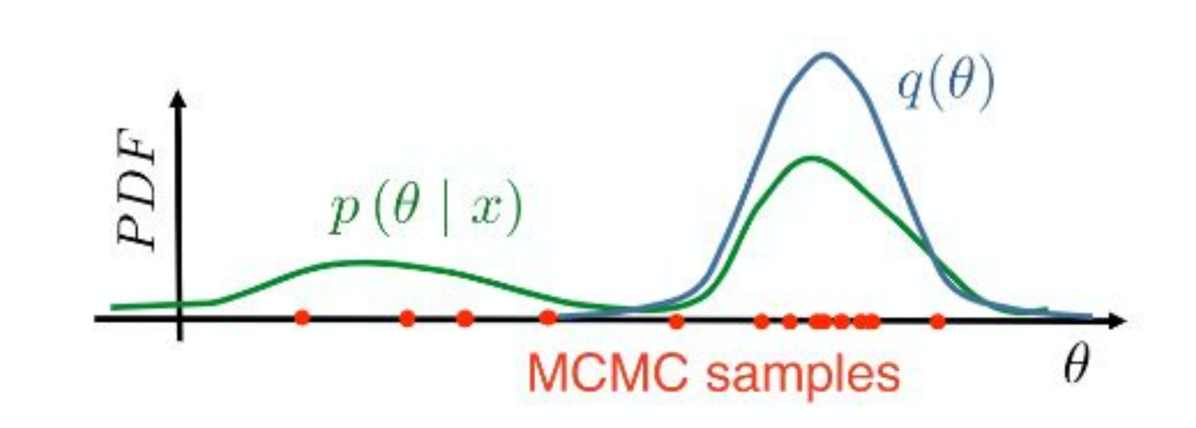
- Biased
- Faster and scalable
- No theoretical guarantees

MCMC

Sample from unnormalized $p(\theta|x)$

- Unbiased
- Needs lots of samples
- Theoretical guarantee of convergence

VI vs MCMC



Importance Sampling (again)

Integral problem -> $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$

<- The marginalized likelihood p(x|Θ) is what we would like to find after getting rid of the latent variables

Introduce new distribution -> $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})\frac{q(\mathbf{z})}{q(\mathbf{z})}d\mathbf{z}$

Re-weight/re-group -> $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}q(\mathbf{z})d\mathbf{z}$

Transformed Integral (after importance sampling) -> $p(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z})}\left[p(\mathbf{x}|\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}\right]$

IS to Variational Inference

Importance weight -> $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}q(\mathbf{z})d\mathbf{z}$

Approximate posterior

Using Jensen's inequality -> $\log p(\mathbf{x}) \geq \int q(\mathbf{z}) \log \left(p(\mathbf{x}|\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}\right) d\mathbf{z}$

$$= \int q(\mathbf{z}) \log p(\mathbf{x}|\mathbf{z}) - \int q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z})}$$

$$\log \int p(x)g(x)dx \geq \int p(x) \log g(x) dx$$

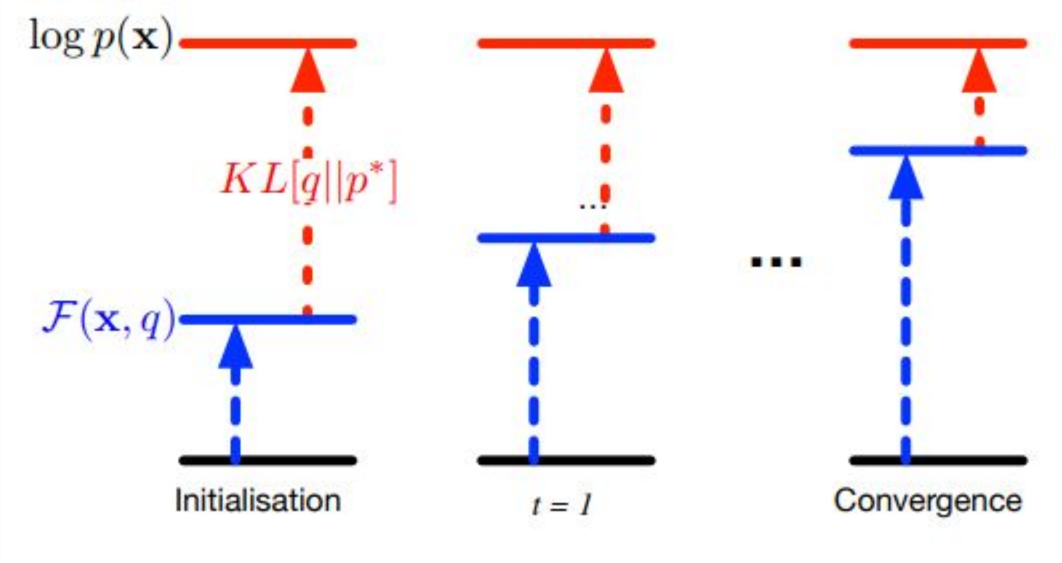
VARIATIONAL LOWER BOUND/ ->
(EVIDENCE LOWER BOUND) ELBO

$$\mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x}|\mathbf{z})] - KL[q(\mathbf{z})||p(\mathbf{z})]$$

(Reconstruction term)
Given z we can generate x:
How good is z at generating x?
(averaged over q)

Kullback-Liebler divergence
(Penalty/Regularizer term)
How different is q(z) to the original prior p(z): don't wanna steer too far

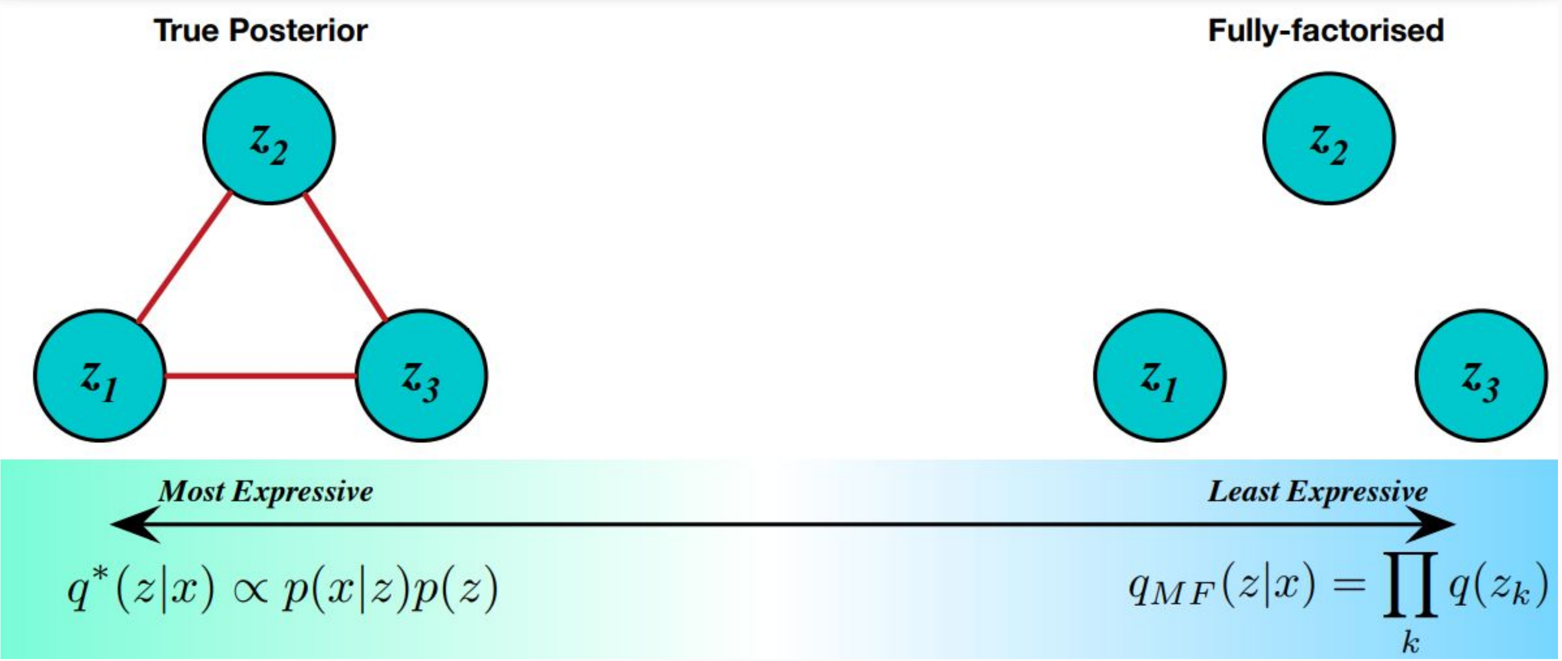
Integral problem -> Optimization problem



Why Variational Inference?

Disadvantages	Advantages
- Approximate posterior ONLY: not guaranteed to find exact in limit - Difficult to optimize - can get stuck in local optima - Underestimates variance of posterior and can bias MLE - Limited theory/guarantees	- Applicable to almost all types of models - Integration -> Optimization - Easy convergence assessment (check if ELBO stops increasing) - Principled and scalable approach for model selection - Faster to converge + Numerically stable - Can use modern architectures (GPUs)

Choosing q



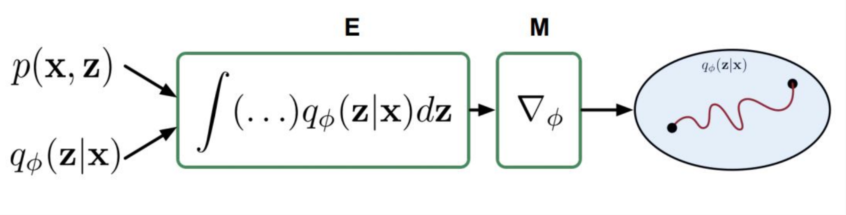
True Posterior
Fully-factorised
z_2
z_1
z_3
z_2
z_1
z_3
Most Expressive
Least Expressive
$q^*(z|x) \propto p(x|z)p(z)$
$q_{MF}(z|x) = \prod_k q(z_k)$

Variational Optimisation

- Variational EM
- Stochastic Variational Inference
- Doubly Stochastic Variational Inference
- Amortized Inference

EM -> Variational EM

<table>
<tr><th>The E-M Algorithm</th><th>Variational E-M</th></tr>
<tr><td>



E-Step: Compute model evidence (expectation over latent variables)

M-Step: Calculate gradients of model parameters from evidence and perform gradient step

</td><td>

$$\mathcal{F}(\mathbf{x}, q) = \mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x}|\mathbf{z})] - KL[q(\mathbf{z})\|p(\mathbf{z})]$$

E-Step: Calculate the gradient wrt variational parameters (instead of calculating the integral we are optimizing the ELBO)

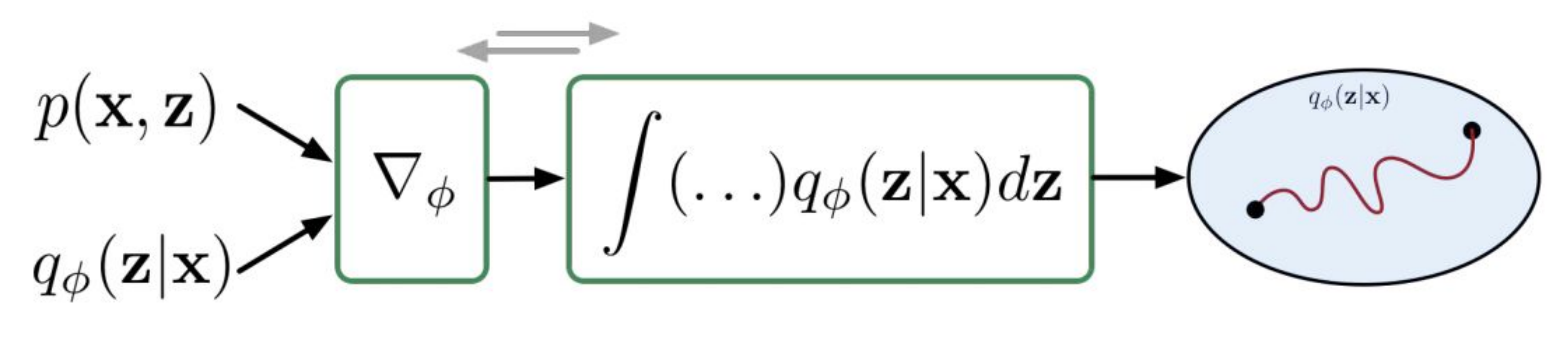
M-Step: Calculate the gradient wrt model parameters

E-step $\phi \propto \nabla_\phi \mathcal{F}(\mathbf{x}, q)$ *Var. params*

M-step $\theta \propto \nabla_\theta \mathcal{F}(\mathbf{x}, q)$ *Model params*

</td></tr>
</table>

Stochastic Inference



Switch* the order of integration-differentiation in the E-M algorithm

$$\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z})} [f_\theta(\mathbf{z})] = \nabla \int q_\phi(\mathbf{z}) | f_\theta(\mathbf{z}) d\mathbf{z}$$

Stochastic Optimization

$$\nabla_{\phi}\mathbb{E}_{q_{\phi}(\mathbf{z})}[f_{\theta}(\mathbf{z})] = \nabla \int q_{\phi}(\mathbf{z}) f_{\theta}(\mathbf{z}) d\mathbf{z}$$

Doubly stochastic estimators:

Pathwise Estimators When f is differentiable and easy to use transformation available	**Score-function estimator** When f is non-differentiable (e.g. discrete latent variable) and q(z) easy to sample from
$= \mathbb{E}_{p(\epsilon)}[\nabla_{\phi} f_{\theta}(g(\epsilon, \phi))]$ $z \sim q_{\phi}(\mathbf{z})$ $\mathbf{z} = g(\epsilon, \phi) \quad \epsilon \sim p(\epsilon)$	$= \mathbb{E}_{q(z)}[f_{\theta}(\mathbf{z}) \nabla_{\phi} \log q_{\phi}(\mathbf{z}))]$

Variational Inference using Pyro

Please follow this Google Colab link

https://drive.google.com/file/d/1mr2U1EMov7l7GFJVcdFWdNDYf41odROJ/view?usp=sharing

File -> Save a copy in Drive = create your own copy to work with

Deep Learning case study: Variational Autoencoders

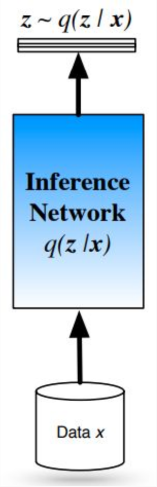
Amortized Inference

$$\phi_n \propto \nabla_\phi \mathbb{E}_{q_\phi(z)}\left[\log p_\theta\left(\mathbf{x}_n | z_n\right)\right] - \nabla_\phi KL\left[q\left(z_n\right) || p(z)\right]$$

How E-step will work?

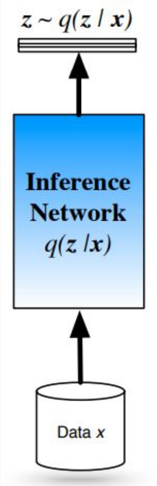
- For each observation: calculate gradients of variational parameters -> optimize the Variational parameters for each observation

Instead of repeatedly calculating the variational gradients for every observation:
We can amortise using a model



Amortized Inference (contd)

Inference network/Encoder/Inverse model: Parameters of q are now a set of global parameters



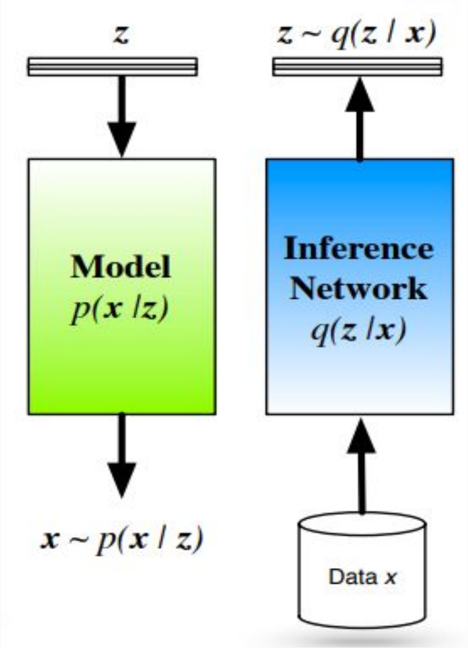
Both model and variational parameters are now global variables
-> can be optimized jointly

Variational Autoencoder

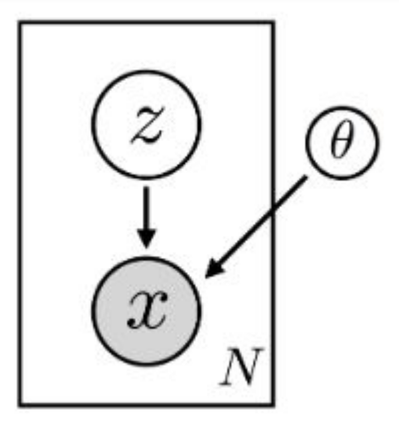
Already seen!

Variational Autoencoder is the combination of:

- **Model:** Latent-Variable model
- **Inference:** Variational Inference
- **Algorithm:** Inference networks +
  Stochastic encoder-decoder



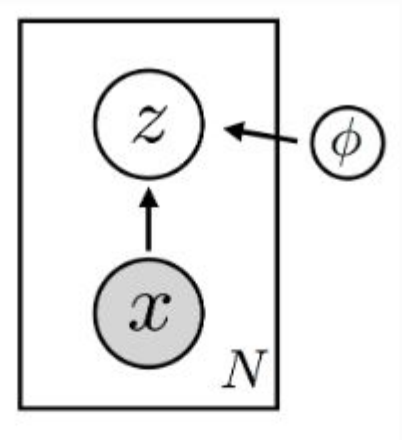
The latent variable model



The model

- N data points $\{x_i\}$
- Each datapoint generated by local latent r.v. z_i
- θ is a parameter (global since all data points depend on it)
- Each x_i depends on z_i in a complex, non-linear way - parameterized by a neural network θ

The inference network



The guide

- Classic VI: Have variational parameters $\{\lambda_i\}$ for each datapoint x_i
- Amortized VI: Instead of variational parameters $\{\lambda_i\}$, learn a function that maps each x_i to an appropriate λ_i. Since we want this mapping to be flexible -> use a neural net

Variational AutoEncoders using Pyro

Please follow this Google Colab link

https://drive.google.com/file/d/1ZqngmNb5bT1TSMjBwH6U1ImE0-aKj7KL/view?usp=sharing

File -> Save a copy in Drive = create your own copy to work with

Final note on PPLs

Bayesian inference is the most-widely used application of PPLs but they are not limited to it. Many (including Pyro) provide methods for causal inference. A lot of them provide modules for applications like forecasting etc.

Outside of statistics and AI, several applications of these languages are in cognitive science and physics.

Ideas from programming languages like effect handling, static analysis, termination checking, program synthesis etc are being ported to PPLs as well.

Resources: PPLs

Courses:

- Frank Wood, UBC, Probabilistic Programming (more about applications)
- Noah Goodman, Stanford, The Design and Implementation of Probabilistic Programming Languages (more about the development of the languages itself)

PPLs:

1. Pyro
2. WebPPL
3. Edward (now Tensorflow Probability)
4. Stan (in multiple languages R/Julia/Python)
5. PyMC

Resources: PPLs

Talks (introductory):

1. Dustin Tran: "What might deep learners learn from probabilistic programming?"
2. Stuart Russell: "Probabilistic programming and AI"
3. "An Overview of Probabilistic Programming" by Vikash K. Mansinghka

Talks (research):

The inaugural International Conference on Probabilistic Programming (PROBPROG) held in 2018 has made its talks available on its youtube channel

Thank You!