On Computational and Probabilistic Inference

Rajat Mani Thomas

Objectives:

- Revisiting *Bayesian inference*.
- A look at Likelihood and the prior
- Probabilistic programming: **Automating Bayesian (like) Inference**
- Probabilistic Toolkit: **(i) Markov Chain Monte Carlo,**

  (ii) Variational inference
- Deep probabilistic models

What is the main problem we are trying to solve ?

Inference

$$\theta$$

What is the main problem we are trying to solve ?

Inference

$$p(\theta)$$

What is the main problem we are trying to solve ?

Inference

$$p(\mathcal{D}|\theta)p(\theta)$$

What is the main problem we are trying to solve ?

Inference

$$P(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$$

What is the main problem we are trying to solve ?

Inference

$$P(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$$

Posterior Likelihood Prior

What is the main problem we are trying to solve ?

Inference

$$P(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$$

Posterior

Likelihood

Prior

The Likelihood function $\mathcal{L}(\theta|x)$

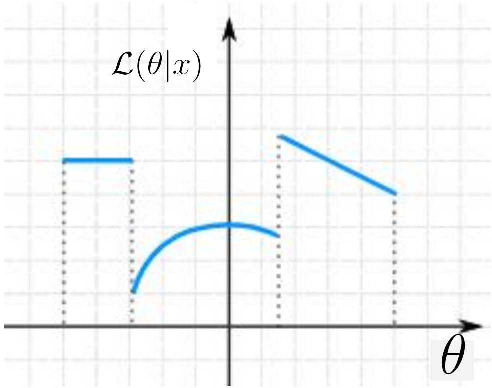
- Generative model of the Data: ***Think Simulations***
- The plausibility of a given parameter in generating a particular outcome.
- Scoring function.

What has now appeared is that the mathematical concept of probability is ... inadequate to express our mental confidence or [lack of confidence] in making ... inferences, and that the mathematical quantity which usually appears to be appropriate for measuring our order of preference among different possible populations does not in fact obey the laws of probability. To distinguish it from probability, I have used the term "**likelihood**" to designate this quantity....

— R. A. Fisher, Statistical Methods for Research Workers [2]

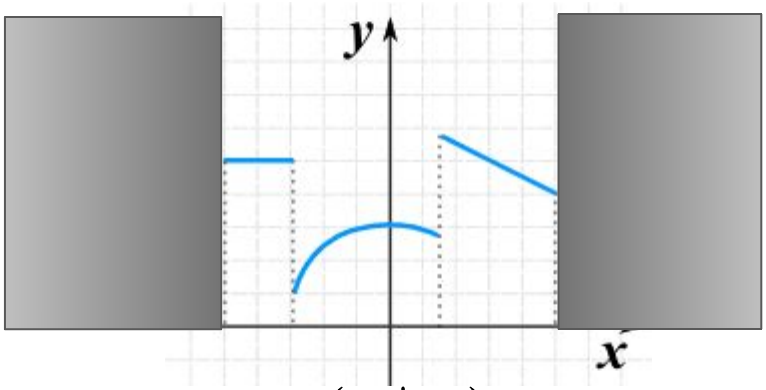
Prior $p(\theta)$

Prior as a searchlight

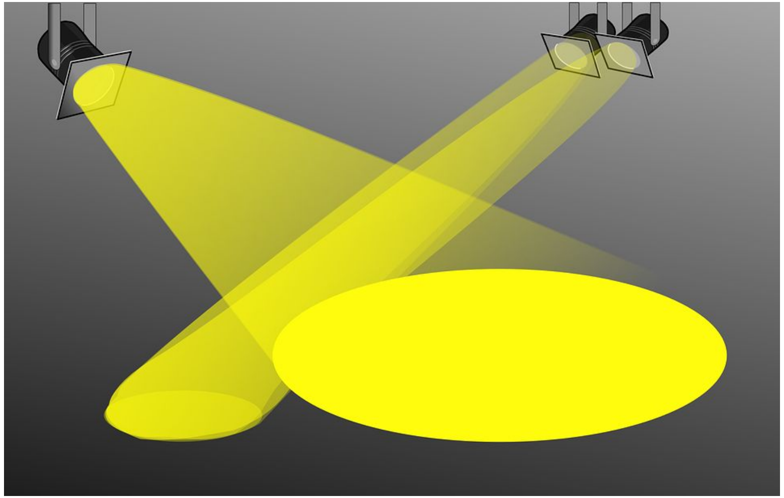


Prior $p(\theta)$

Prior as a searchlight



$$\mathcal{L}(\theta|x)$$



The two (hypothesized) ways to *Intelligence*

Deep Learning

Programs with R.V. and Probabilistic calculations

The two (hypothesized) ways to *Intelligence*

Deep Learning

$$\frac{\partial \ \mathrm{OUTPUT}}{\partial \left(\cdots \right)}$$

```python
from torch import nn
import torch.nn.functional as F
import torch

class RTprofile2paramConv(nn.Module):

    def __init__(self, sizeOutputLayer):

        super(RTprofile2paramConv, self).__init__()

        self.conv1 = nn.Conv1d(2, 16, 13, stride=1)
        self.conv2 = nn.Conv1d(16, 32, 6, stride=1)
        self.conv3 = nn.Conv1d(32, 64, 3, stride=1)
        self.avgpool = nn.AdaptiveAvgPool1d((1))
        self.linear1 = nn.Linear(64, 256)
        self.linear2 = nn.Linear(256, 64)
        self.linear3 = nn.Linear(64, sizeOutputLayer)

    def forward(self, x):

        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = self.avgpool(x)
```

Programs with R.V. and Probabilistic calculations

The two (hypothesized) ways to *Intelligence*

Deep Learning

$$\frac{\partial \ \text{OUTPUT}}{\partial \left(\text{code} \right)}$$

```python
from torch import nn
import torch.nn.functional as F
import torch

class RTprofile2paramConv(nn.Module):

    def __init__(self, sizeOutputLayer):

        super(RTprofile2paramConv, self).__init__()

        self.conv1 = nn.Conv1d(2, 16, 13, stride=1)
        self.conv2 = nn.Conv1d(16, 32, 6, stride=1)
        self.conv3 = nn.Conv1d(32, 64, 3, stride=1)
        self.avgpool = nn.AdaptiveAvgPool1d((1))
        self.linear1 = nn.Linear(64, 256)
        self.linear2 = nn.Linear(256, 64)
        self.linear3 = nn.Linear(64, sizeOutputLayer)

    def forward(self, x):

        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = self.avgpool(x)
```

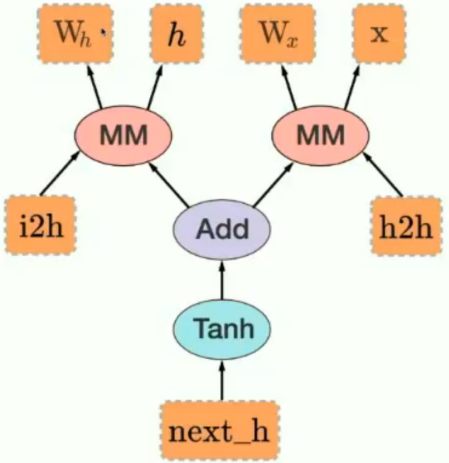
PyTorch Autograd

```python
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
next_h = i2h + h2h
next_h = next_h.tanh()

next_h.backward(torch.ones(1, 20))
```



The two (hypothesized) ways to *Intelligence*

$$P(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$$

Programs with R.V. and Probabilistic calculations

The two (hypothesized) ways to *Intelligence*

$$P(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$$

Programs with R.V. and Probabilistic calculations

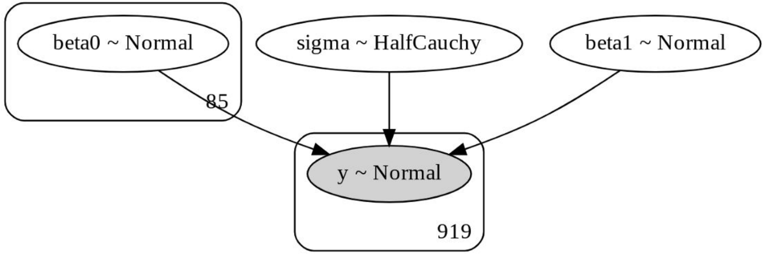
In [12]:
```
with Model() as unpooled_model:

    beta0 = Normal('beta0', 0, sd=1e5, shape=counties)
    beta1 = Normal('beta1', 0, sd=1e5)
    sigma = HalfCauchy('sigma', 5)

    theta = beta0[county] + beta1*floor

    y = Normal('y', theta, sd=sigma, observed=log_radon)
model_to_graphviz(unpooled_model)
```

Out[12]:



The case for *probabilistic programming languages*

Democratize model building

and *Inference*

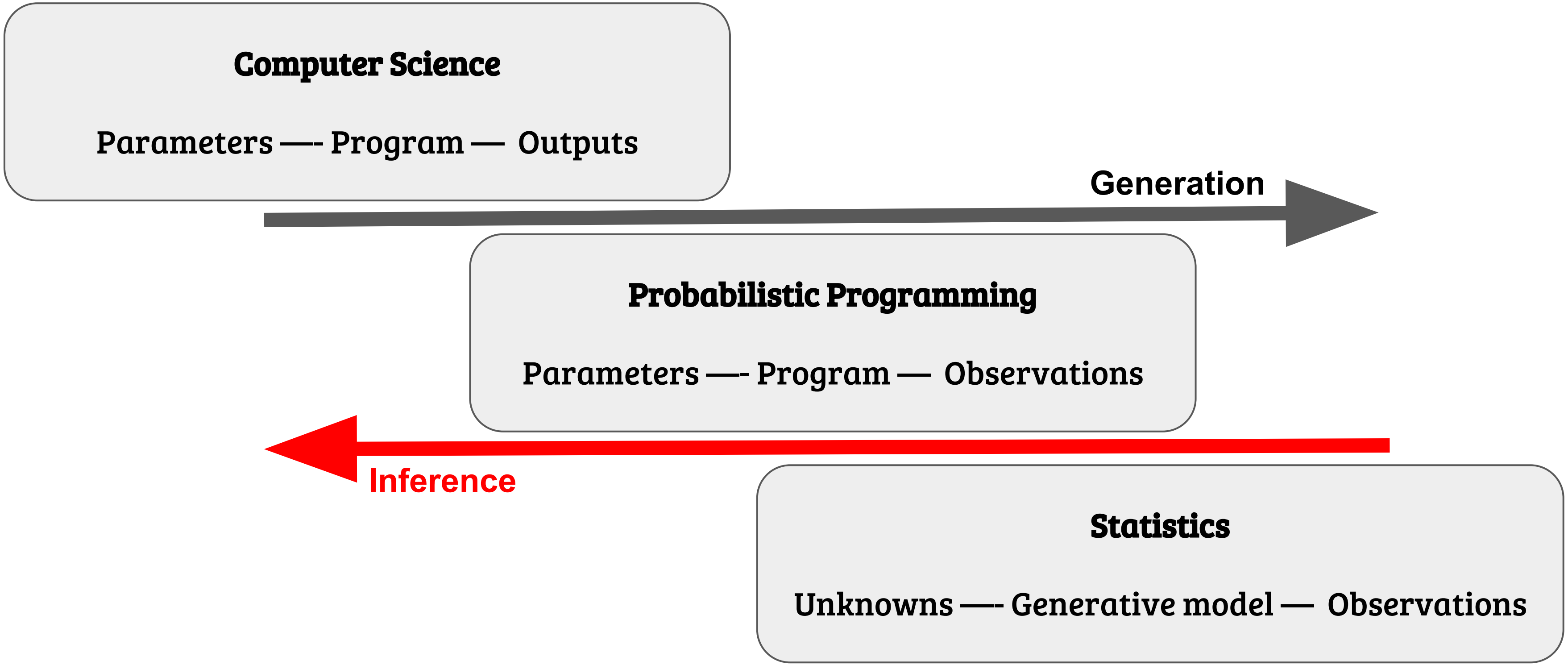
$$P(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}$$

What is a probabilistic program?

Any imperative or functional program with two additional constructs:

1. Ability to draw values at random from distributions $z \sim p(z)$
2. Ability to condition variables with observations $P(z|\mathcal{D})$

PP - A CS way of doing Statistical Inference



An example of a PP

$$x := 0; y := 0; W = 1$$

```
x=sample(beta(3,2));

if (sample(flip(x))) {

  y=sample(normal(x*x,1));

} else {

  y=sample(normal(5*x,1));

}
obs(normal(y,1),3);
```

An example of a PP

```
x=sample(beta(3,2));

if (sample(flip(x))) {

  y=sample(normal(x*x,1));

} else {

  y=sample(normal(5*x,1));

}
obs(normal(y,1),3);
```

$$x := 0; y := 0; W = 1$$

$$x := 0.4; y := 0; W = 1$$

An example of a PP

```
x=sample(beta(3,2));

if (sample(flip(x))) {

  y=sample(normal(x*x,1));

} else {

  y=sample(normal(5*x,1));

}
obs(normal(y,1),3);
```

$$x := 0; y := 0; W = 1$$

$$x := 0.4; y := 0; W = 1$$

$$x := 0.4; y := 0.1; W = 1$$

An example of a PP

```
x=sample(beta(3,2));
if (sample(flip(x))) {
  y=sample(normal(x*x,1));
} else {
  y=sample(normal(5*x,1));
}
obs(normal(y,1),3);
```

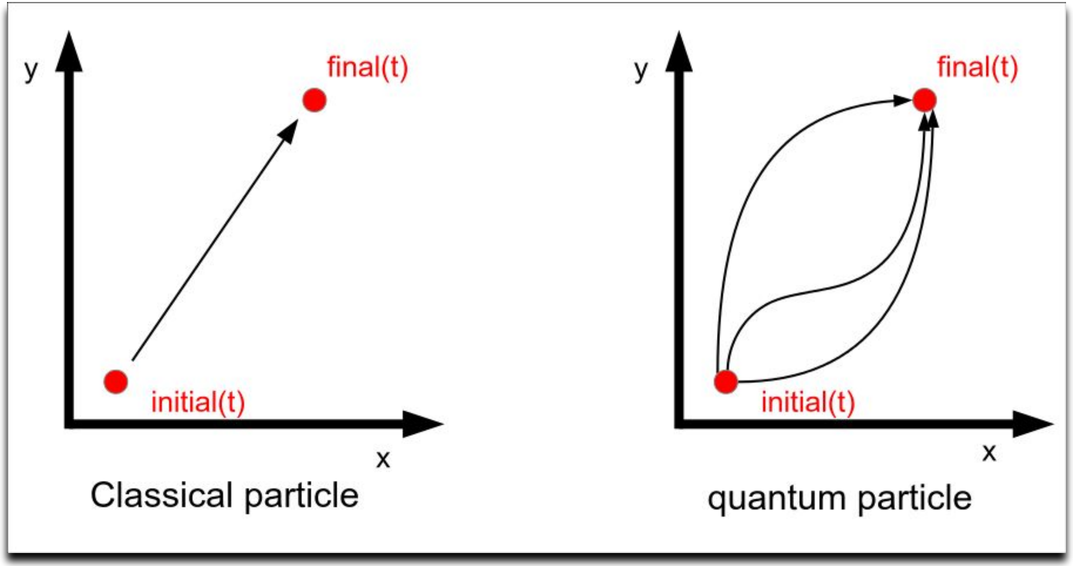
$x := 0; y := 0; W = 1$

$x := 0.4; y := 0; W = 1$

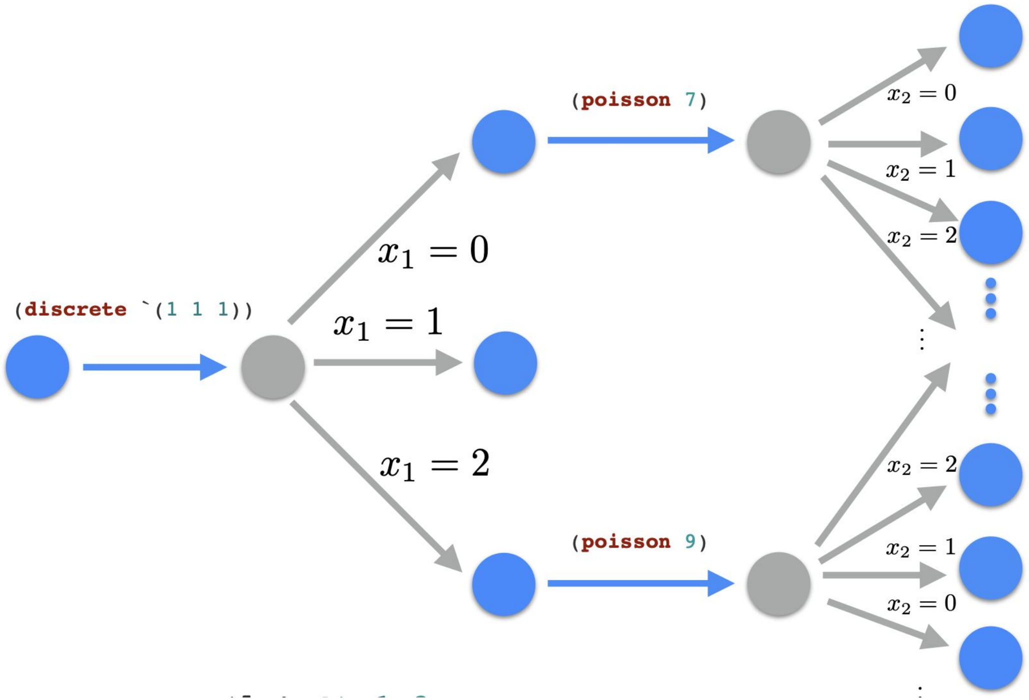
$x := 0.4; y := 0.1; W = 1$

$x := 0.4; y := 0.1; W = 0.01$

Program traces

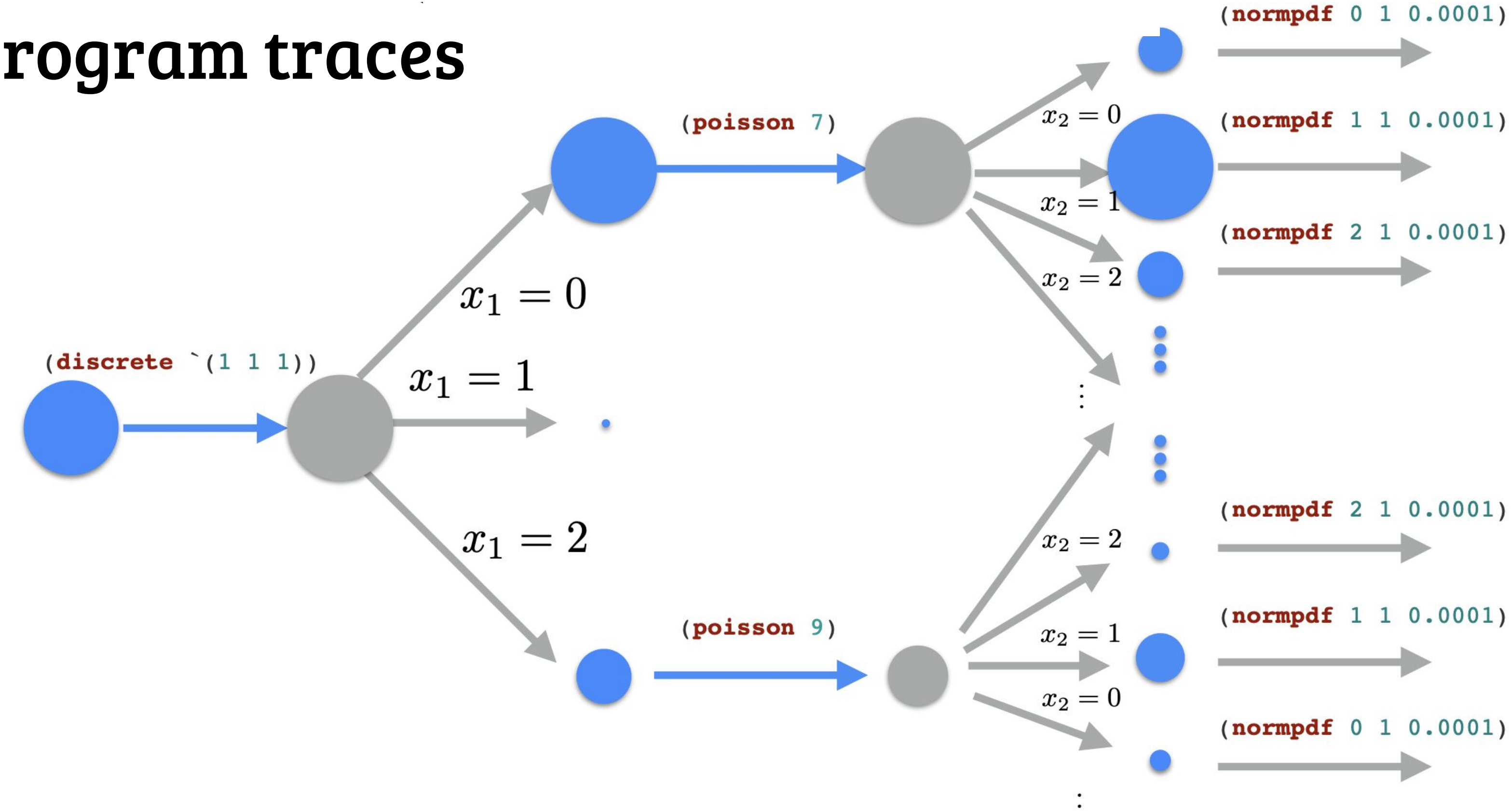


Program traces



F.wood (NIPS, 2015)

Program traces



F.wood (NIPS, 2015)

Getting to the posterior

$$p(\tau)p(\mathcal{D}|\tau)$$

Getting to the posterior

$$p(\tau)p(\mathcal{D}|\tau)$$

Getting to the posterior

$$p(\tau)p(\mathcal{D}|\tau)$$

Getting to the posterior : Importance Sampling

1. Run the program N times generate traces:

$$p(\tau|\mathcal{D}) = \frac{p(\tau)p(\mathcal{D}|\tau)}{Z}$$

2. Approximate the posterior:

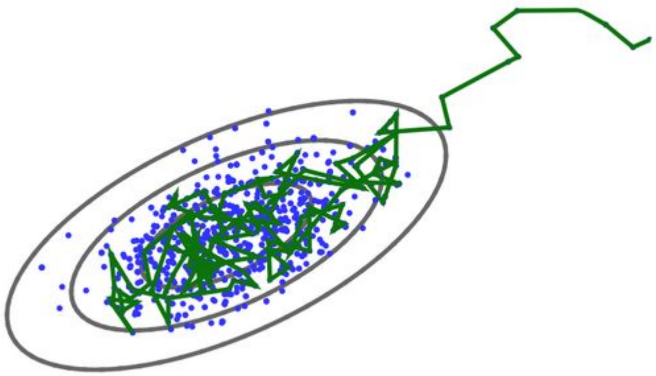
$$(\tau_1, w_1), (\tau_2, w_2) \ldots (\tau_N, w_N)$$

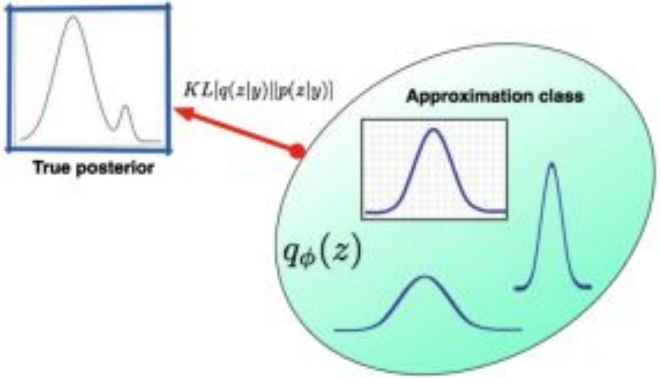
$$p(\tau|D) \approx \frac{\sum_i \tau_i w_i}{\sum_i w_i}$$

Algorithms that make it feasible

MCMC

Variational Inference





Markov Chain Monte Carlo

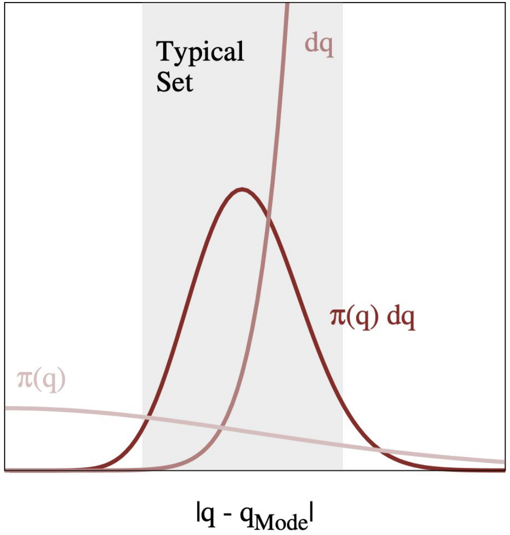
Assumes very little...

Can you run your program and generate samples? - SIMULATION

Can you calculate (even an un-normalized) density of observation? - SCORE

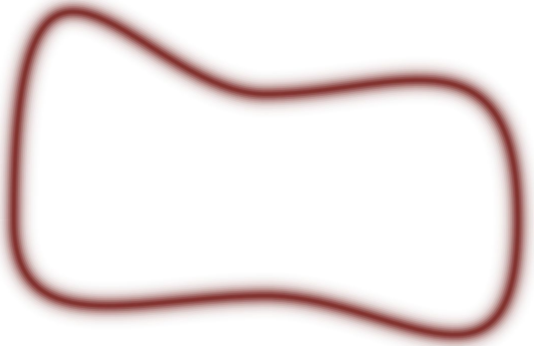
Markov Chain Monte Carlo - 101

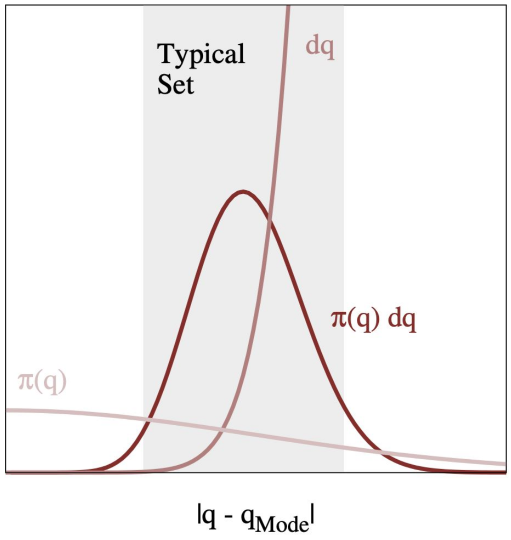
$$\mathbb{E}_\pi[f] = \int_\mathcal{Q} \mathrm{d}q \, \pi(q) \, f(q) \, .$$



Markov Chain Monte Carlo - 101

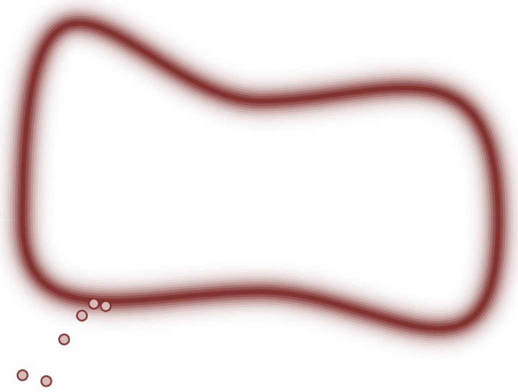
$$\mathbb{E}_\pi[f] = \int_{\mathcal{Q}} \mathrm{d}q\, \pi(q)\, f(q)\,.$$

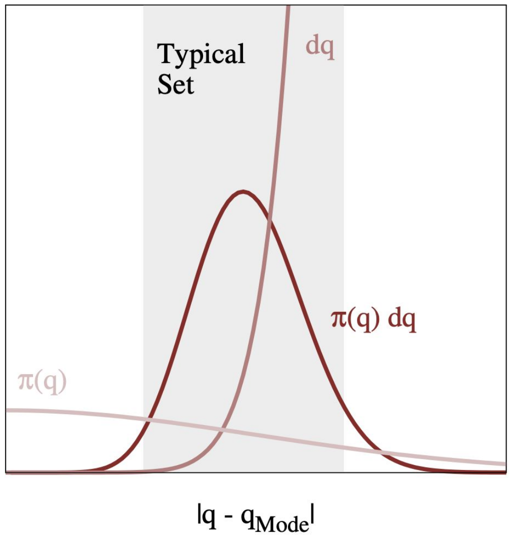




Markov Chain Monte Carlo - 101

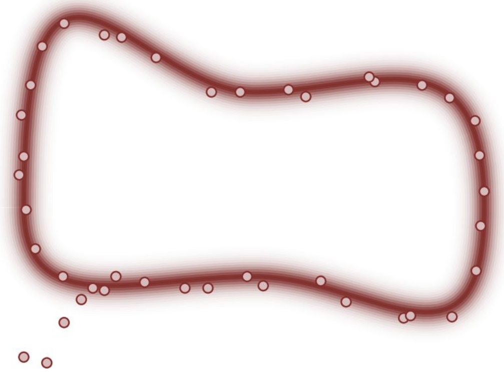
$$\mathbb{E}_\pi[f] = \int_{\mathcal{Q}} \mathrm{d}q\, \pi(q)\, f(q)\,.$$

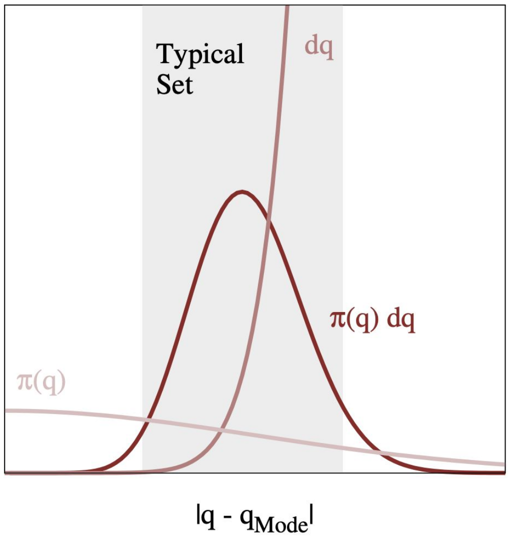




Markov Chain Monte Carlo - 101

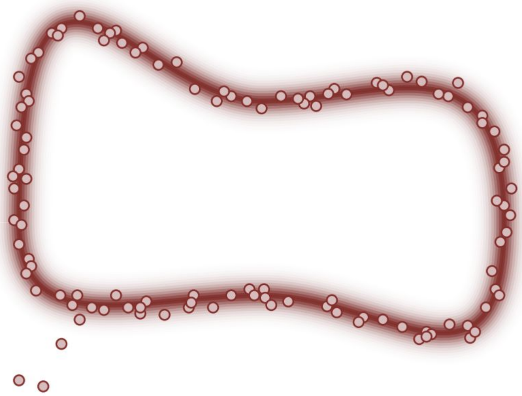
$$\mathbb{E}_{\pi}[f] = \int_{\mathcal{Q}} \mathrm{d}q\, \pi(q)\, f(q)\,.$$

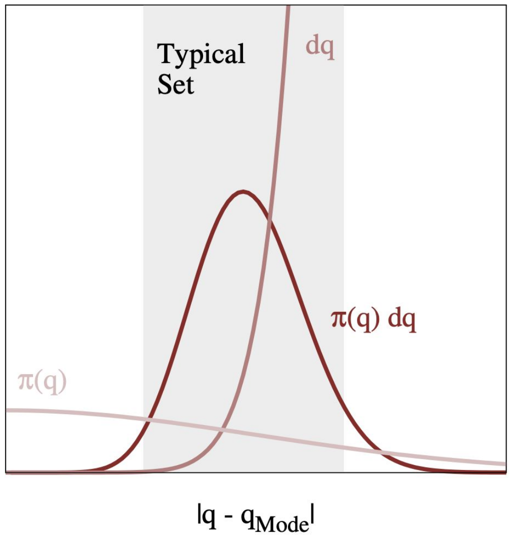




Markov Chain Monte Carlo - 101

$$\mathbb{E}_\pi[f] = \int_{\mathcal{Q}} \mathrm{d}q\, \pi(q)\, f(q)\,.$$





Markov Chain Monte Carlo - 101

$$\mathbb{E}_\pi[f] = \int_{\mathcal{Q}} \mathrm{d}q \, \pi(q) \, f(q) \, .$$

$$\hat{f}_N = \frac{1}{N} \sum_{n=0}^{N} f(q_n) \, .$$

$$\lim_{N \to \infty} \hat{f}_N = \mathbb{E}_\pi[f] \, .$$

MCMC - 101 : Metropolis-Hastings

$$a(q' \mid q) = \min\left(1, \frac{\mathbb{Q}(q \mid q')\,\pi(q')}{\mathbb{Q}(q' \mid q)\,\pi(q)}\right).$$

$$\mathbb{Q}(q' \mid q) = \mathcal{N}(q' \mid q, \Sigma),$$

$$a(q' \mid q) = \min\left(1, \frac{\pi(q')}{\pi(q)}\right).$$

Variational Inference

1. Choose a "nice" family of distributions

$$p(\tau|\mathcal{D}) = \frac{p(\tau)p(\mathcal{D}|\tau)}{Z}$$

2. Cast inference as an optimization problem

$$\{q_\theta(\tau)\}$$

$$\theta = \mathrm{argmax}_\theta KL\{q_\theta(\tau)||p(\tau|\mathcal{D})\}$$

Variational Inference

1. Choose a "nice" family of distributions

$$p(\tau|\mathcal{D}) = \frac{p(\tau)p(\mathcal{D}|\tau)}{Z}$$

2. Cast inference as an optimization problem

$$\{q_\theta(\tau)\}$$

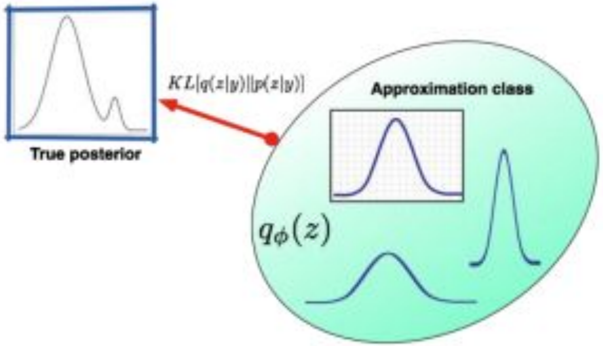
$$\theta = \mathrm{argmax}_\theta KL\{q_\theta(\tau)||p(\tau|\mathcal{D})\}$$

Variational Inference

$$KL\{q_\theta(\tau)||p(\tau|\mathcal{D})\} := \mathbb{E}_{q_\theta(\tau)}\left[\log \frac{q_\theta(\tau)}{p(\tau|\mathcal{D})}\right]$$

$$:= \mathbb{E}_{q_\theta(\tau)}\left[\log \frac{q_\theta(\tau)p(\mathcal{D})}{p(\tau,\mathcal{D})}\right]$$

$$:= p(\mathcal{D}) - \mathbb{E}_{q_\theta(\tau)}\left[\log \frac{p(\tau,\mathcal{D})}{q_\theta(\tau)}\right]$$



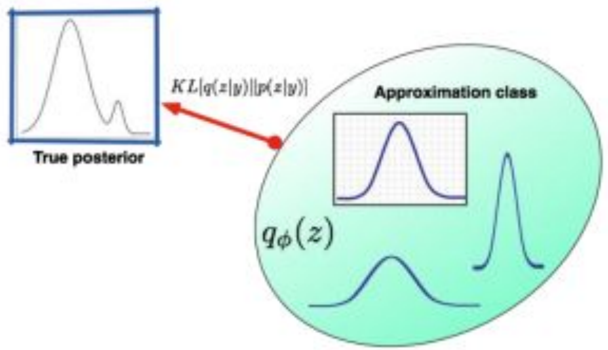
Variational Inference

$$KL\{q_\theta(\tau)||p(\tau|\mathcal{D})\} := \mathbb{E}_{q_\theta(\tau)}\left[\log \frac{q_\theta(\tau)}{p(\tau|\mathcal{D})}\right]$$

$$:= \mathbb{E}_{q_\theta(\tau)}\left[\log \frac{q_\theta(\tau)p(\mathcal{D})}{p(\tau,\mathcal{D})}\right]$$

$$:= p(\mathcal{D}) - \mathbb{E}_{q_\theta(\tau)}\left[\log \frac{p(\tau,\mathcal{D})}{q_\theta(\tau)}\right]$$

$$p(\tau,\mathcal{D}) = p(\tau)p(\mathcal{D}|\tau)$$



Advantages

Amortized inference:

Easily expressive language for model

Write your Generative model, PP takes care of the inference

What would it look like?

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

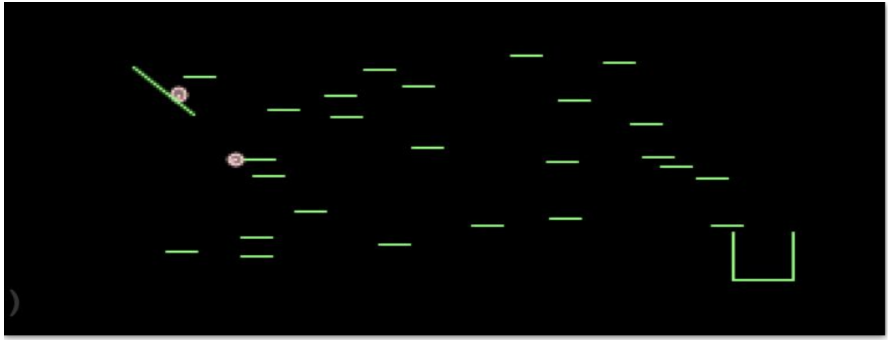
x

Simulation

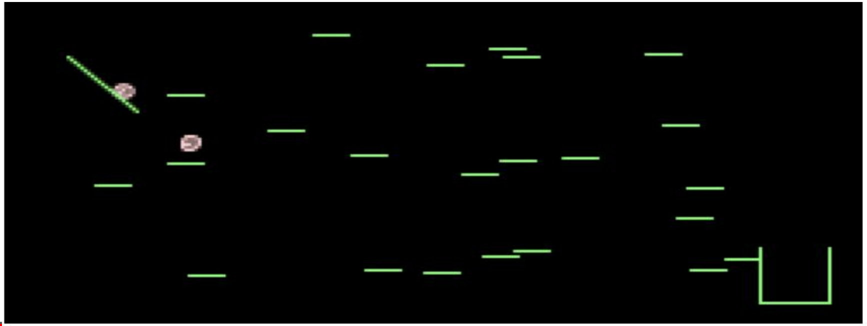
y

Power spectra

Examples of what PP can do

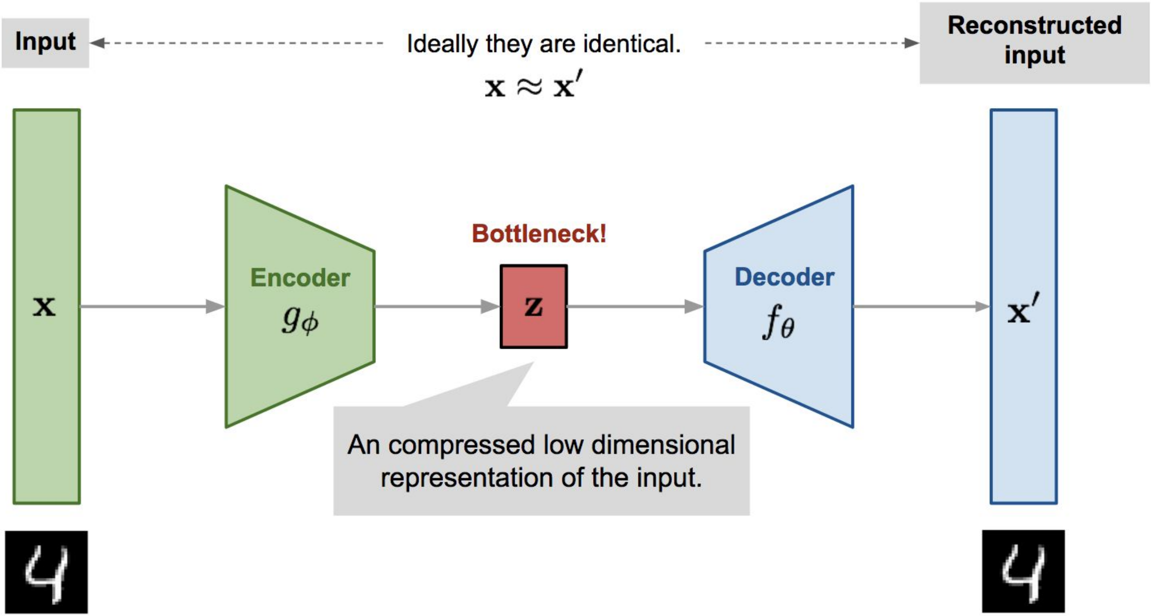


Construct a world in which 20% of balls go into the basket



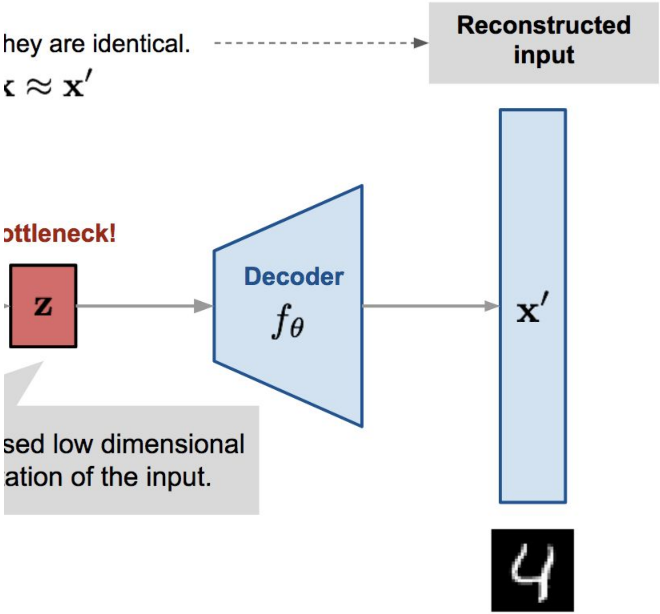
Deep Learning + Probabilistic Programming

An example of the Variational Autoencoder

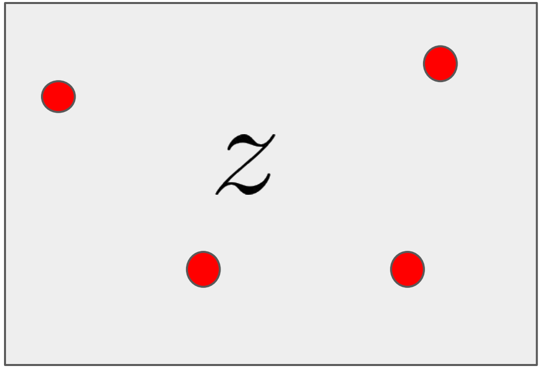


Deep Learning + Probabilistic Programming

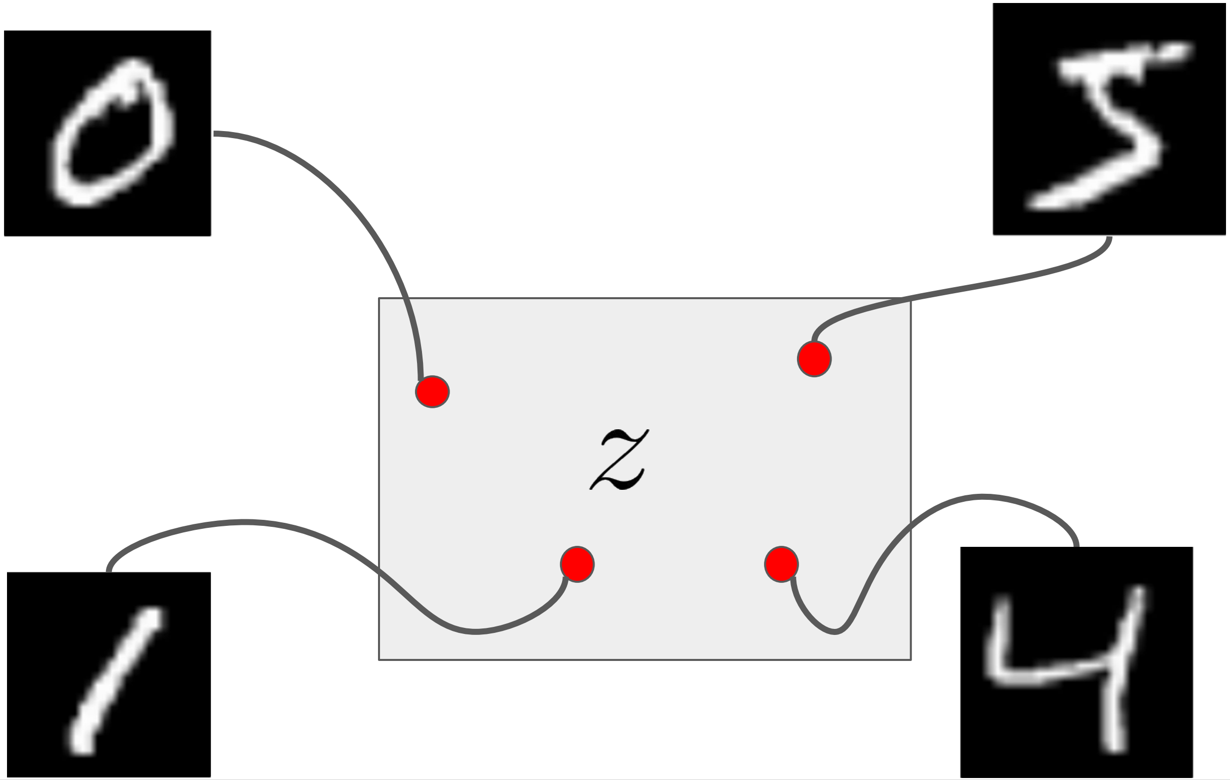
An example of the Variational Autoencoder



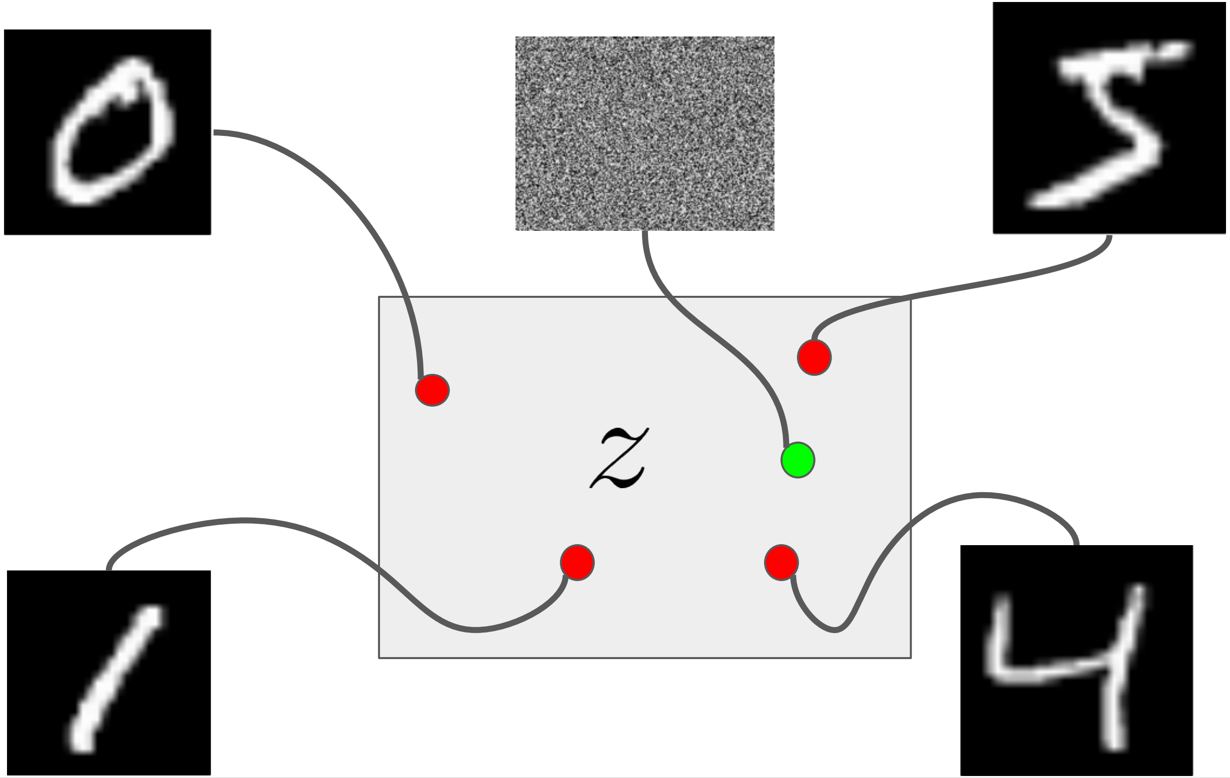
Canonical Autoencoder



Canonical Autoencoder



Canonical Autoencoder

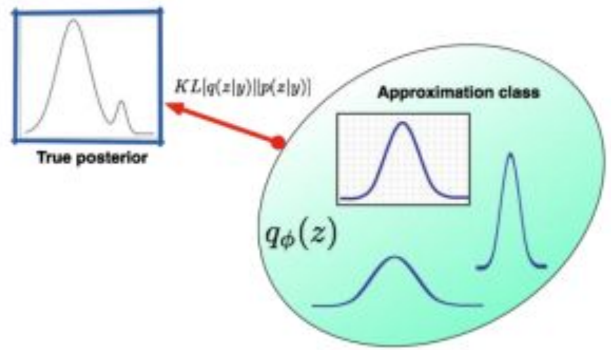


Variational Inference

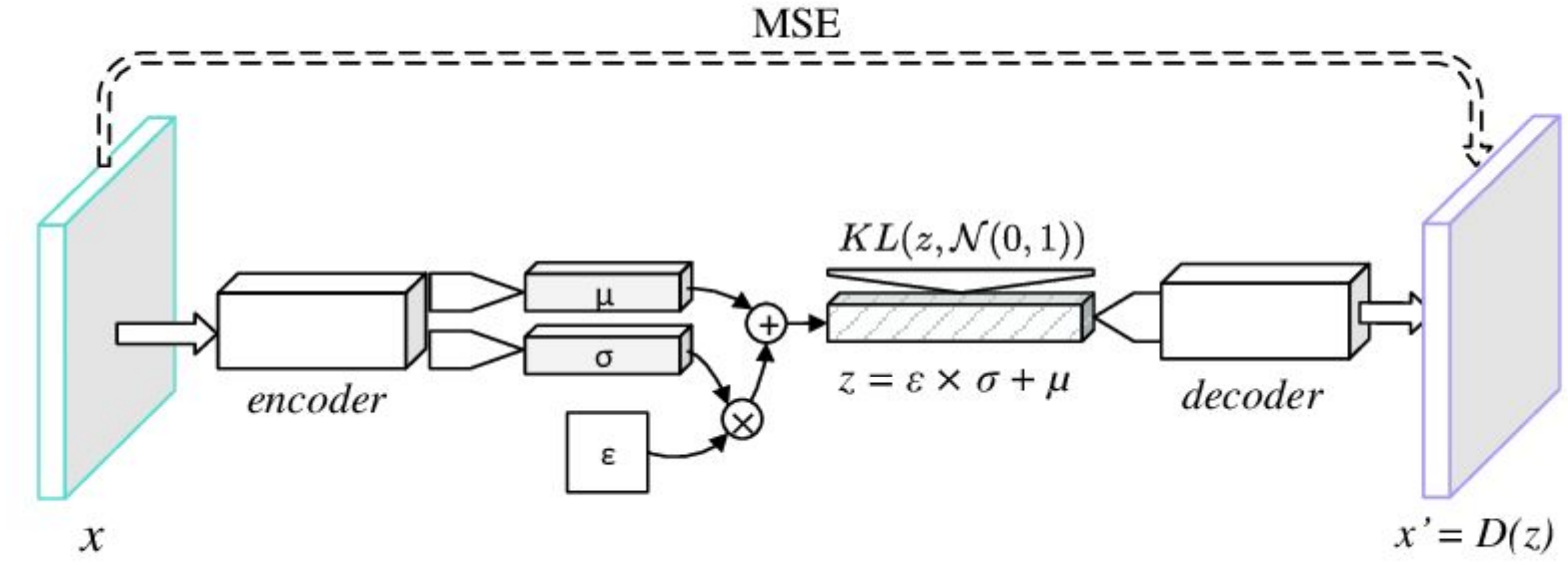
$$KL\{q_\theta(\tau)||p(\tau|\mathcal{D})\} := \mathbb{E}_{q_\theta(\tau)}\left[\log\frac{q_\theta(\tau)}{p(\tau|\mathcal{D})}\right]$$

$$:= \mathbb{E}_{q_\theta(\tau)}\left[\log\frac{q_\theta(\tau)p(\mathcal{D})}{p(\tau,\mathcal{D})}\right]$$

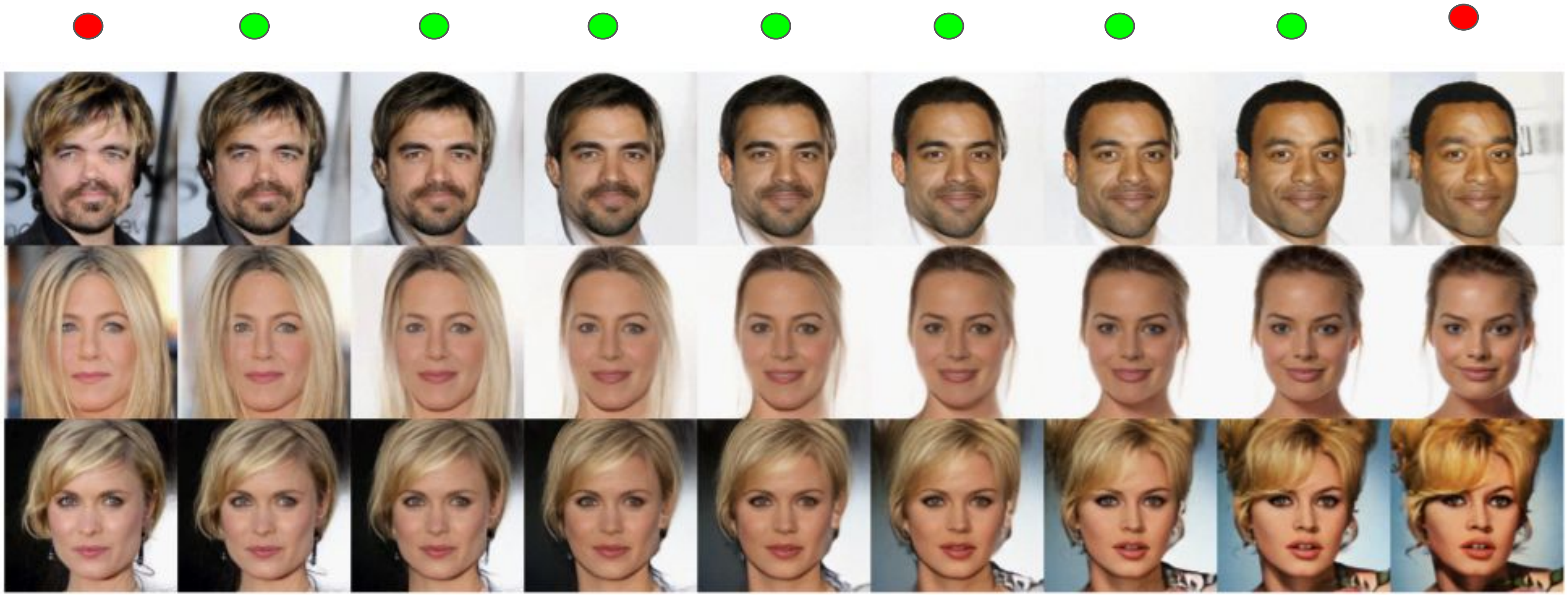
$$:= p(\mathcal{D}) - \mathbb{E}_{q_\theta(\tau)}\left[\log\frac{p(\tau,\mathcal{D})}{q_\theta(\tau)}\right]$$



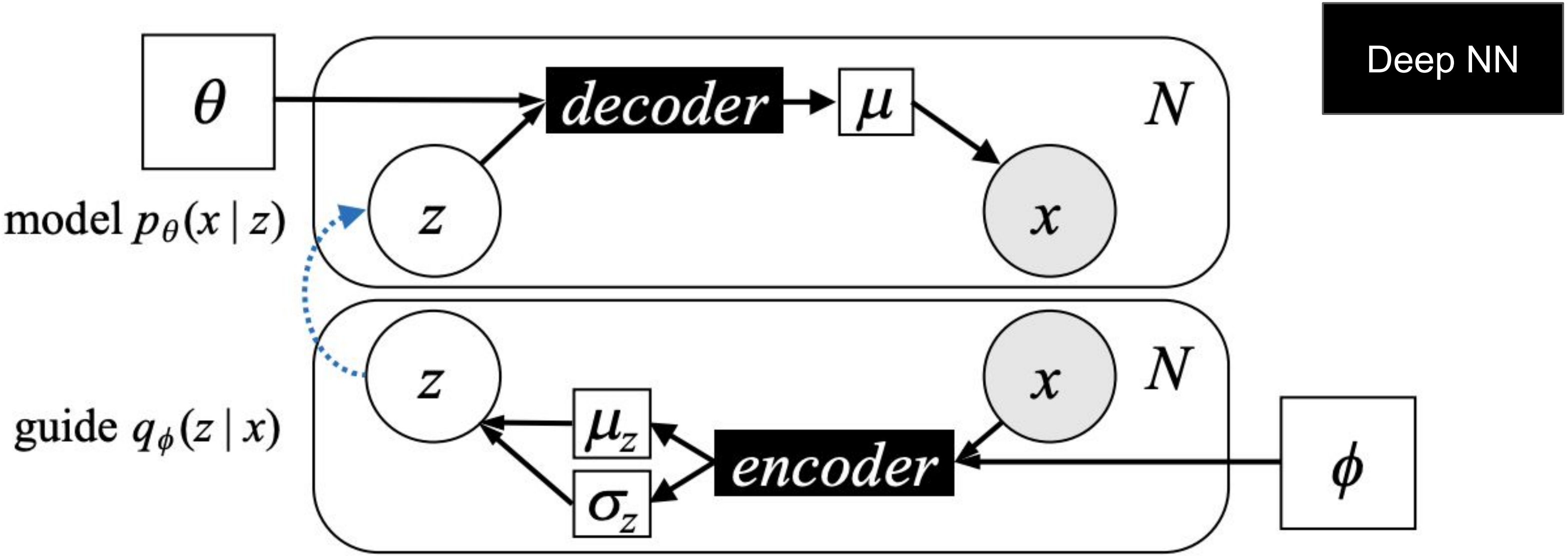
Variational Autoencoder



Variational Autoencoder



Variational Autoencoder



Deep *Mixed* Probabilistic Models

A new paradigm for model building

Deep *Mixed* Probabilistic Models

A new paradigm for model building

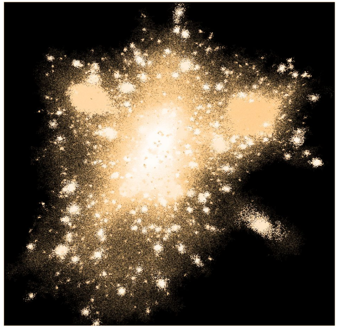
$$p_\theta(\mathcal{D}|z)$$

Deep *Mixed* Probabilistic Models

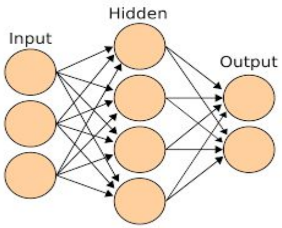
A new paradigm for model building

$$p_\theta(\mathcal{D}|z)$$

$\theta \rightarrow \{$



,



$\}$

Conclusions

1. Probabilistic programming languages are making it easy to run inference on anything that can be written as a computer code

2. Advances in MCMC techniques like Hamiltonian Monte Carlo and Variational Inference are the workhorses of inference algorithms

3. Mixed programming paradigms will be the way forward

References:

Probabilistic Programming Language: ————————- Frank Wood (NIPS Tutorial on Probabilistic Programming),

Probabilistic Programming Applications: ————————- Josh Tenenbaum (MIT), Noah Goodman (Stanford)

MCMC/HMC: ————————————— Michael Betancourt, https://arxiv.org/pdf/1701.02434.pdf

Variational Inference: ————————- Max Welling, Dirk Kingma, Danilo Rezende, David Blei

Frameworks for (Deep) Probabilistic Programming:

Stan, TFP (Google, Tensorflow), Pyro (UBER, pytorch), Pyprob (Atilim, Frank Wood's lab), Probabilistic C, …