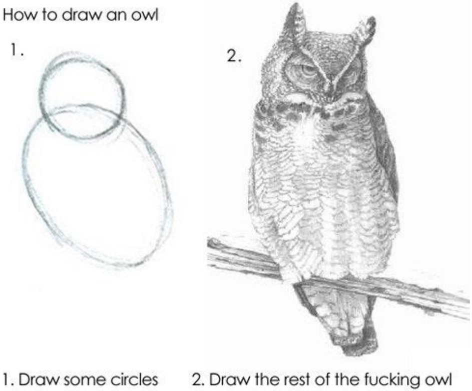
An introduction to bayesian data analysis with PyMC3

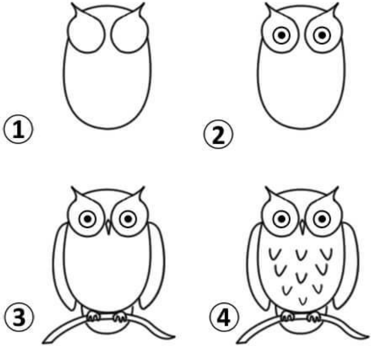
Sean Meling Murray and Solveig Masvie

inmeta

17. februar 2020

Googling bayesian analysis vs this presentation





¿¿¿¿¿¿¿ updates

Road map

1. Theory
   (a) The basics of Bayesianism
   (b) Markov chain Monte Carlo methods (MCMC)
2. Practice
   (a) Probabilistic programming with PyMC3

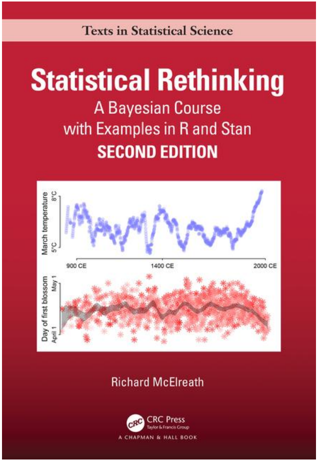
¡¡¡¡¡¡¡ HEAD =======
¿¿¿¿¿¿¿ updates

What is Bayesian data analysis?

"A Bayesian is one who, vaguely expecting a horse, and catching a glimpse of a donkey, strongly believes he has seen a mule."

It's just counting!

- Richard McElreath: "Bayesian inference is just counting."
- Count all the ways observed data could have arisen according to assumptions
- Assumptions that can arise in more ways are more consistent with the data, and therefore more plausible

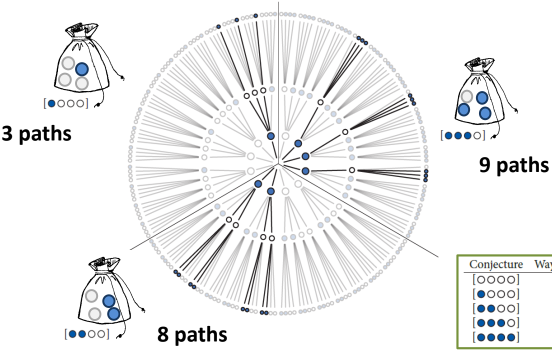


The Garden of Forking Data

Counting possibilities

Evidence (i.e. data): ● ○ ●

Draw 3 marbles from the bag (with replacement)



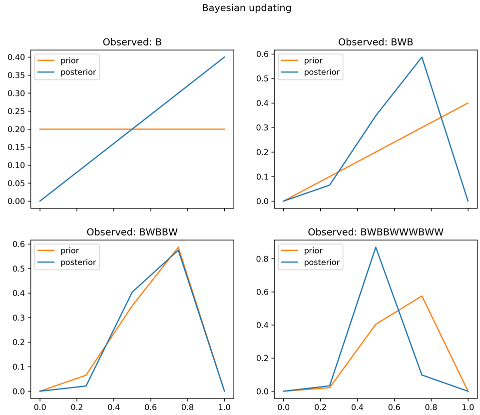
Conjecture	Ways to produce ●○●
[○○○○]	$0 \times 4 \times 0 = 0$
[●○○○]	$1 \times 3 \times 1 = 3$
[●●○○]	$2 \times 2 \times 2 = 8$
[●●●○]	$3 \times 1 \times 3 = 9$
[●●●●]	$4 \times 0 \times 4 = 0$

Priors, posteriors and likelihoods

Prior $p(\theta)$: Encodes assumptions about θ
Likelihood $p(y\,|\,\theta)$: How were the observed data generated?
Posterior $p(\theta\,|\,y)$: Assumptions about θ consistent with data

$$p(\theta|y) = \frac{p(\theta)p(y|\theta)}{\int p(y|\theta)p(\theta)\,d\theta}$$
$$\propto p(\theta)p(y|\theta)$$

$$\textit{posterior} \propto \textit{prior} \cdot \textit{likelihood}$$



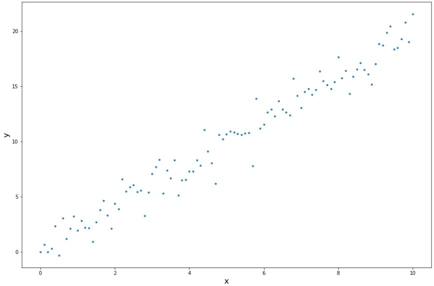
Model

- In the Bayesian analysis we are trying to describe the process that generated the data.
- We assume that the process can be expressed as a parametic model

$$M(input, \theta)$$

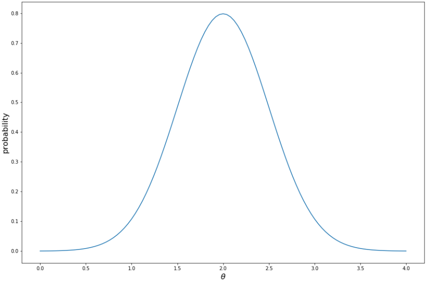
Example

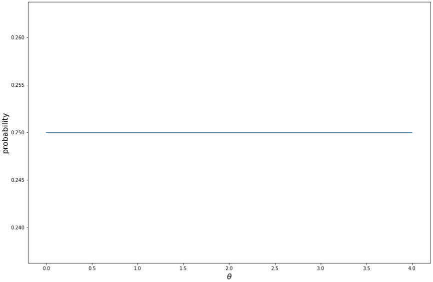
The oh so boring linear regression example....



Model: $y_i = \theta x_i + \varepsilon_i, \quad \varepsilon \sim N(0, \sigma^2)$

Example: Priors

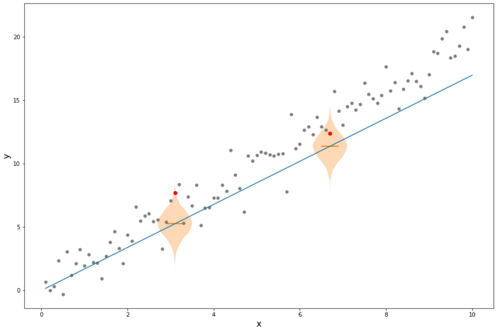




Example: Likelihood

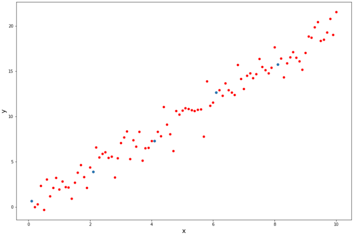
$$p(y|\theta) = \prod_{i=1}^{n} p(y_i|\theta)$$

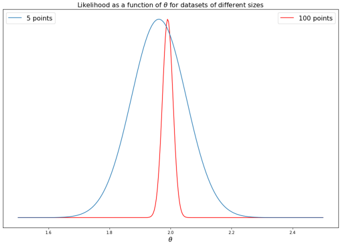
$$p(y_i|\theta) = N(y_i|\theta x_i, \sigma^2)$$

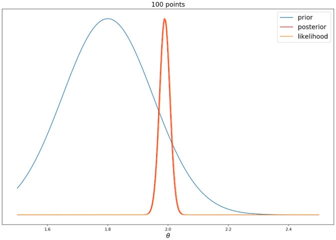


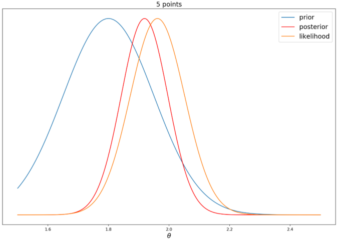
What is the probability of my data given the model and parameters I've chosen?

Example: Size of dataset









inmeta

Towards a Bayesian workflow

1. Design the model (tell the data story)
   (a) Describe data generating process:
   $p(y\,|\,\theta) = \theta^{\sum_i \mathbf{1}(y_i=1)}(1-\theta)^{\sum_i \mathbf{1}(y_i\neq 1)}$
   (b) Encode assumptions about parameters: $p(\theta) = \frac{1}{5}$
2. Condition on the data (update step)
   (a) $p(\theta\,|\,y) \propto p(\theta)p(y\,|\,\theta)$
3. Evaluate the model (critique), and either
   (a) be happy, it worked as planned, or
   (b) return to step 1

The Garden of Forking Data, revisited

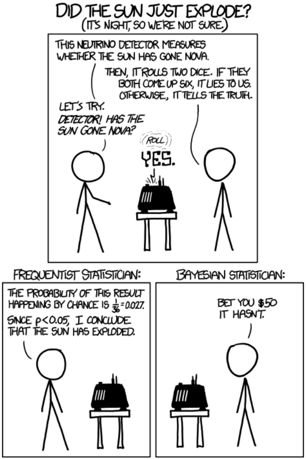
```python
import numpy as np

## Observed data
data = np.array([1, 0, 1])

## Conjectures (possible values for theta)
# array([0., 0.25, 0.5, 0.75, 1.])
theta = np.linspace(0, 1, num=5)

# Prior distribution over theta
# array([0.2, 0.2, 0.2, 0.2, 0.2])
uniform_prior = np.repeat(0.2, repeats=5)

# Likelihood function
def likelihood(data, theta):
    num_blue = data.sum()
    num_white = len(data) - num_blue
    return theta**num_blue * (1 - theta)**num_white

# Unnormalised posterior over theta
# array([0., 0.009375, 0.025, 0.028125, 0.])
unnormalised_posterior = uniform_prior * likelihood(data, theta)

# Posterior distribution over theta
# array([0., 0.15, 0.4, 0.45, 0.])
posterior = unnormalised_posterior / unnormalised_posterior.sum()
```

The Garden of Forking Data, revisited

```
import numpy as np

# Number of paths consistent with conjectures
forking_data = np.array([0, 3, 8, 9, 0])

# array([0., 0.15, 0.4, 0.45, 0.]
forking_data / forking_data.sum()
```

The Frequentist vs. Bayesian debacle

- Frequentist statistics
  - Probability = limiting frequency
  - Uncertainty arises from sampling variation
- Bayesian statistics
  - Frequency ≠ probability
  - Uncertainty arises from ignorance
- Intuitive way to represent and take into account uncertainty
- Easily incorporates prior information



But priors are subjective! Buu! Hiss!

- Non-informative vs. informative priors
  - Forking data example: $\mathit{Uniform}(0, 1)$ vs. $\mathit{Uniform}(\frac{1}{4}, \frac{3}{4})$
- Test your assumptions, e.g. using prior predictive checks
- More about priors:
  - Gelman et al. (2017): `https://arxiv.org/abs/1708.07487`
  - Bayesian Methods for Hackers: `https://tinyurl.com/v2yzyqv`
- For more Bayesian challenges, see e.g. Gelman and Yao (2020): `https://tinyurl.com/sbr2tev`

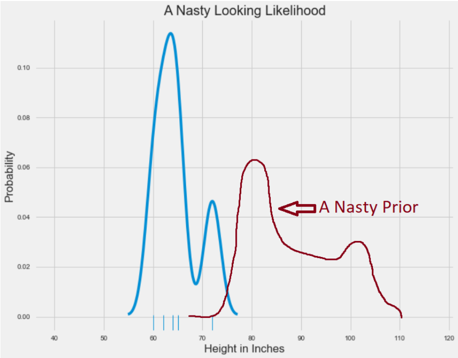
Telling the data story

“ People think in terms of stories - thus the unreasonable power of the anecdote to drive decision-making, well-founded or not. Existing analytics largely fails to provide this kind of story; instead, numbers seemingly appear out of thin air.”

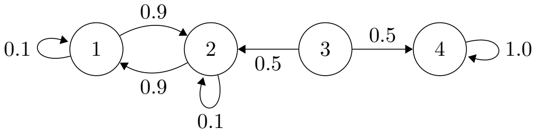
Beau Cronin, *Why Probabilistic Programming Matters (2013)*

How to sample from intractable posteriors?

- In textbooks, nice, we can sample directly from well-behaved, analytical posteriors
- In the real world, often impossible to sample directly from $p(\theta|y)$
- A general class of algorithms called **Markov chain Monte Carlo** let us approximately sample from posterior

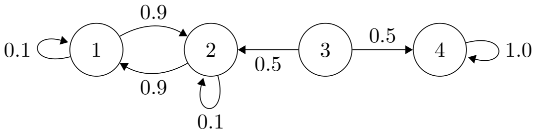


Markov chains



- A Markov chain defines a joint probability distribution over a sequence of events
- Used to model events in discrete and continuous time (manufacturing processes, queuing systems, etc.)
- Typically interested in the long-term distribution of being in a certain state given an initial starting state

Properties of a Markov chain



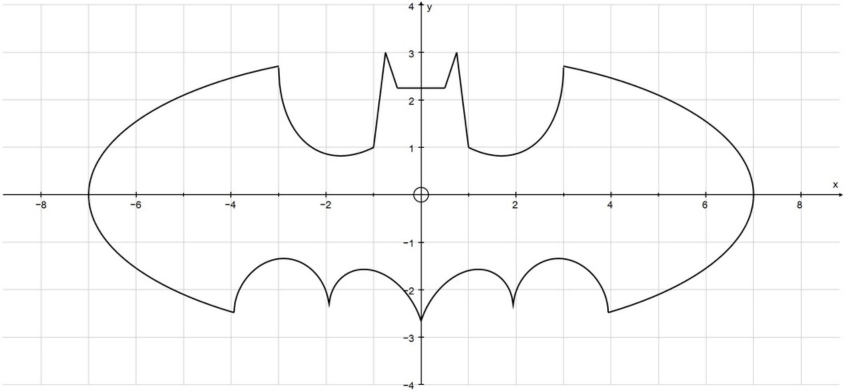
- Memoryless (Markov property): $P(X_t \mid X_{t-1}, X_{t-2}, \ldots, X_1) = P(X_t \mid X_{t-1})$
- Under certain conditions, guaranteed to have a unique, limiting distribution (aka. stationary distribution)
  - Long run proportion of time spent in each state
  - For more details, check `https://github.com/smu095/presentations/`

Monte Carlo simulations

- Monte Carlo (MC) simulations are just a way to approximate numerical results using repeated random sampling
- Main idea:
  - Generate N samples $x_1, \ldots, x_N$ from $p(x)$, approximate f using the empirical distribution of $\{f(x_n)\}_{n=1}^{N}$
  - $\mathbb{E}[f] = \int f(x) p(x)\, dx \approx \frac{1}{N} \sum_{n=1}^{N} f(x_n) = \hat{f}$
- MC estimates converge thanks to Law of Large Numbers
  - $(\hat{f} - \mathbb{E}[f]) \rightarrow \mathcal{N}\left(0, \frac{Var[f]}{N_{eff}}\right)$ as $N \rightarrow \infty$

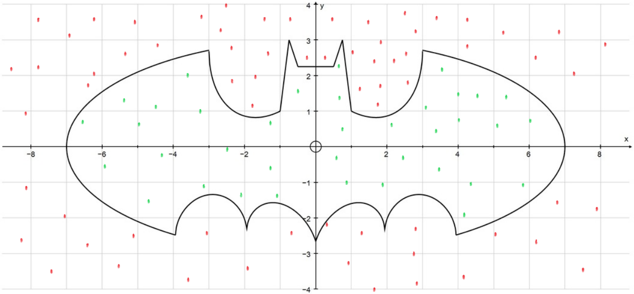
Application: Evaluate integrals

- Calculate difficult integrals, such as the area of the Batman sign

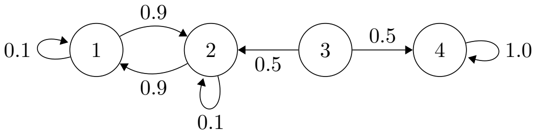


Application: Evaluate integrals

- Repeatedly sample $(u_1, u_2) \sim \text{Uniform}(-\frac{1}{2}, \frac{1}{2})$
- Calculate area A of Batman sign as $A = \text{area of rectangle} \times \frac{\text{green dots}}{\text{all dots}}$



Properties of a Markov chain



- Memoryless (Markov property):
  $P(X_t \mid X_{t-1}, X_{t-2}, \ldots, X_1) = P(X_t \mid X_{t-1})$
- Under certain conditions, guaranteed to have a unique, limiting distribution (aka. stationary distribution)
  - Long run proportion of time spent in each state
  - For more details, check `https://github.com/smu095/presentations/`

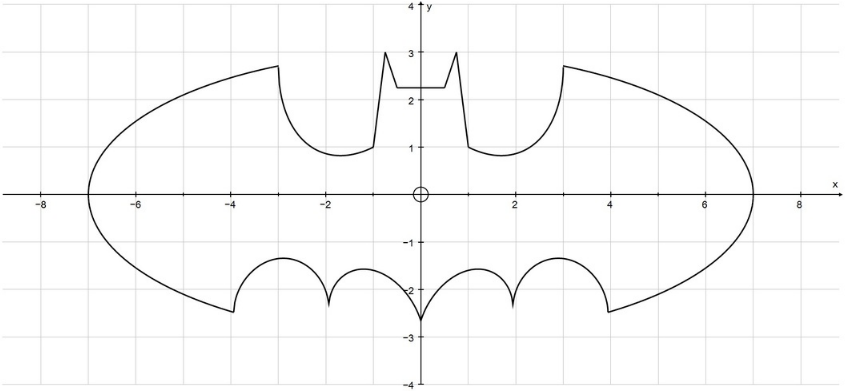
Monte Carlo simulations

- Monte Carlo (MC) simulations are just a way to approximate numerical results using repeated random sampling
- Main idea:
  - Generate N samples $x_1, \ldots, x_N$ from $p(x)$, approximate f using the empirical distribution of $\{f(x_n)\}_{n=1}^{N}$
  - $\mathbb{E}[f] = \int f(x)p(x)\,dx \approx \frac{1}{N}\sum_{n=1}^{N} f(x_n) = \hat{f}$
- MC estimates converge thanks to Law of Large Numbers
  - $(\hat{f} - \mathbb{E}[f]) \to \mathcal{N}\left(0, \frac{Var[f]}{N_{eff}}\right)$ as $N \to \infty$

inmeta

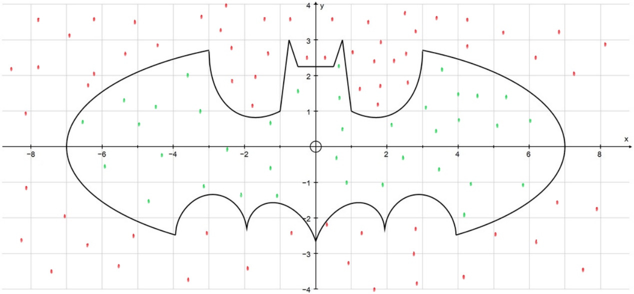
Application: Evaluate integrals

- Calculate difficult integrals, such as the area of the Batman sign



Application: Evaluate integrals

- Repeatedly sample $(u_1, u_2) \sim \text{Uniform}(-\frac{1}{2}, \frac{1}{2})$
- Calculate area A of Batman sign as $A = \text{area of rectangle} \times \frac{\text{green dots}}{\text{all dots}}$



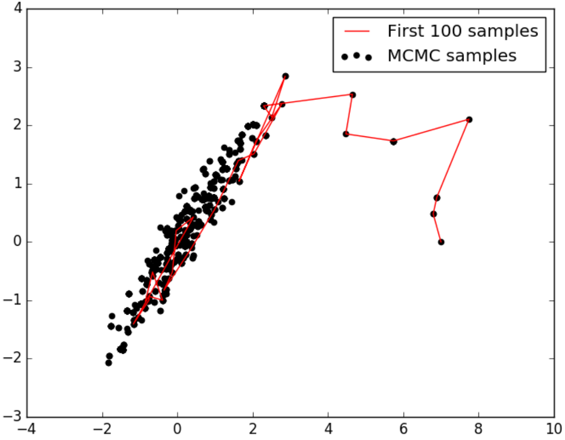
Markov chain Monte Carlo methods

- **Main idea:** Construct a Markov chain over the parameter space where the stationary distribution is the posterior $p(\theta \mid y)$
  - For more details, check `https://github.com/smu095/presentations/`
- Randomly move around the parameter space such that the fraction of time spent at each randomly sampled parameter value is proportional to the true target density $p(\theta \mid y)$.
- Use the sequence of parameters generated by Markov chain to calculate MC approximations of any quantity of interest

Algorithmic view of MCMC

1. Start at current position.
2. Propose new position.
3. Accept/reject new position based on position's adherence to data and prior distributions.
   (a) If reject: Remain in your current position, return to step 1.
   (b) If accept: Move to the new position, return to step 1.
4. After large number of iterations, return sequence of accepted positions.

MCMC illustration



Metropolis algorithm

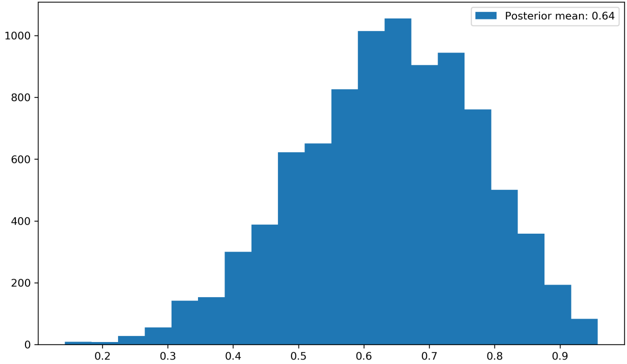
1. Start at some initial parameter value θ_0
2. For $t = 1$ to T:
   (a) Sample θ^* from a proposal distribution $q(\theta^* \,|\, \theta_{t-1})$
   $$\theta^* = \theta_{t-i} + \mathcal{N}(0, \text{jump scale})$$
   (b) Evaluate $\alpha_{\theta^*} = \dfrac{p(\theta^* \,|\, y)}{p(\theta_{t-1} \,|\, y)}$.
   (c) Accept/reject θ^*
   If $\alpha_{\theta^*} \geq 1$ set $\theta_t = \theta^*$
   Else draw $u \sim Uniform(0,1)$ set $\theta_t = \begin{cases} \theta^*, & \text{if } u < \alpha_{\theta^*} \\ \theta_{t-1}, & \text{o.w.} \end{cases}$

inmeta

Metropolis in Python

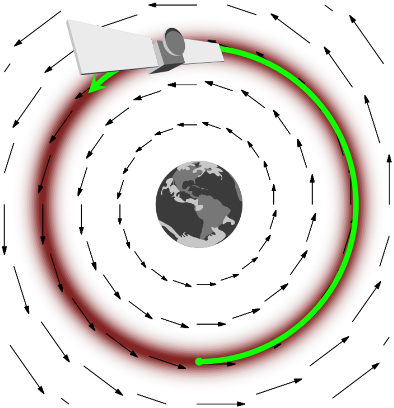
```
"""Metropolis sampling from posterior of mean"""
<<<<<<< HEAD

import numpy as np

# Marbles data
data = np.array([1, 0, 1])

def log_p(theta, x):
    N = len(x)
    k = np.sum(x)
    logprior = np.repeat(0.2, len(x))
    loglik = k*np.log(theta + 10e-8) + (N - k)*np.log(1 - theta + 10e-8)
    return np.sum(logprior + loglik)

# Initial value for theta
theta = 0.5
=======
import numpy as np
data = np.array([1, 0, 1]) # Observations

def log_p(theta, x): # Proportional log posterior
    N, k = len(x), np.sum(x)
    logprior = np.repeat(0.2, N)
    loglik = k*np.log(theta + 10e-8) + (N - k)*np.log(1 - theta + 10e-8)
    return np.sum(logprior + loglik)

theta = 0.5 # Initial value for theta
>>>>>>> updates
```

Metropolis in Python



Sampling algorithms

- Many different sampling algorithms exist
- Currently, the state-of-the-art is Hamiltonian Monte Carlo (HMC)
  - See e.g. Betancourt (2017): `https://arxiv.org/abs/1701.0`



Probabilistic programming in Python

- PyStan
- PyMC3
- Pyro
- TensorFlow Probability

PyMC3

- Python package for for fitting Bayesian models using modern methods (MCMC, variational inference, etc.)
- Well-documented, large suite of statistical models
- Highly flexible, but easy to use
- Uses Theano-backend (not good)
- PyMC4 in development, uses TensorFlow Probability-backend