Probabilistic Programming, Modeling the world through Uncertainty

Student's Scientific Chapter
Computer Engineering Department
Amirkabir University of Technology
(Tehran Polytechnic)

Amirkabir Artificial Intelligence Student Summit Workshop (AAISS)

Ali Izadi

Before we start

- No **Prerequisites**: From basics to advanced methods. I try to explain everything as simple as possible.
- Ask any questions during the workshop.
- All codes and contents are available in https://github.com/aliizadi/probabilistic-programming-workshop
- Workshop = methods and problem description + implementation.
- Getting introduced to probabilistic programming, I hope you enjoy and start studying and use it. I promise I mention books and courses to continue.

Agenda

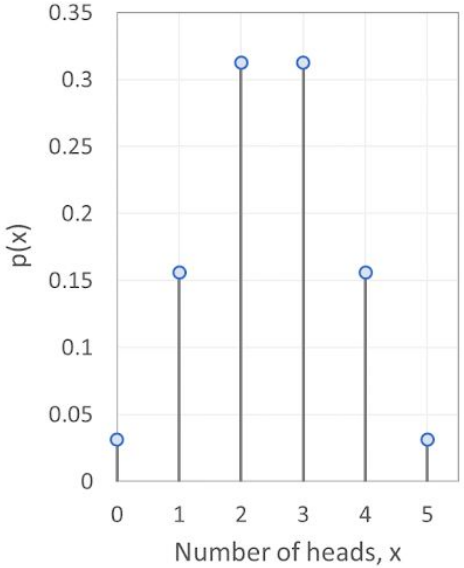
- Why we need uncertainty and how to address it using probabilistic programming?

- Bayesian thinking (updating our beliefs):
  1- model problem with probability distributions
  2- find best models with probabilistic programming.
  3- describe uncertainty in your results because of uncertainty in problem.

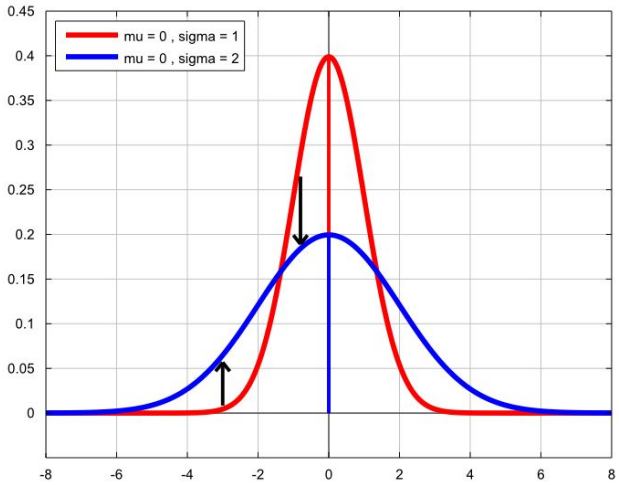
- Introduction to PyMC3 and Tensorflow Probability

Probability Distributions (our first tools for describing uncertainty using **random variables**)

PMF (discrete random variable)



PDF (continuous random variable)



Normal distribution

How to Draw?

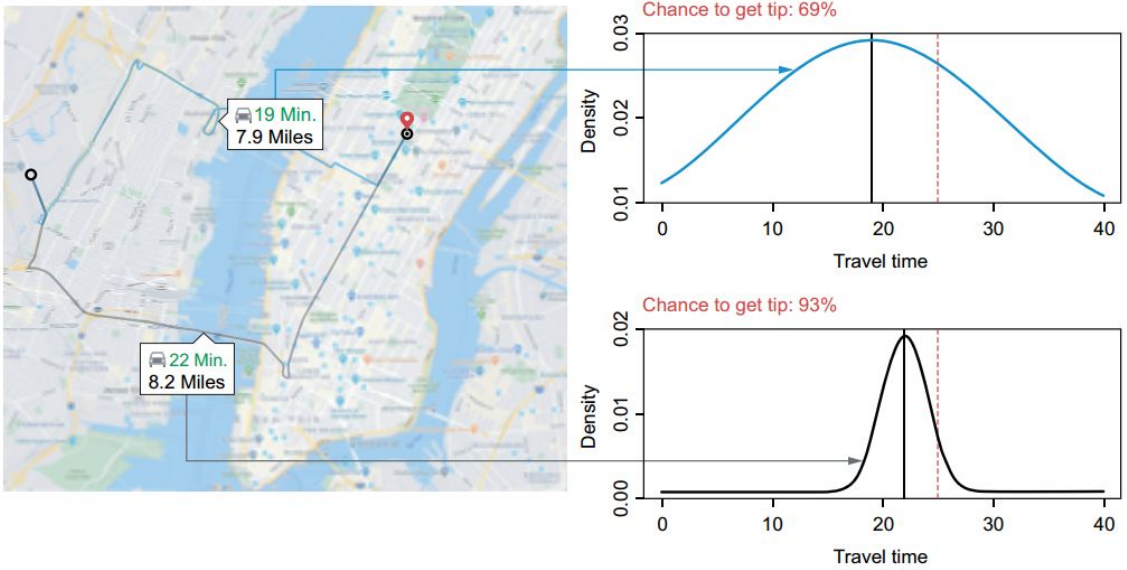
1- histogram:
plt.hist(data)

2- probability density function:
For example normal distribution:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Why we need modeling uncertainty?

Example: predict travel time for a taxi driver. Tip if arrive in 25 minutes



Two routes:

- If you don't consider uncertainty first one is better. 19 min < 22 min
- If consider second is better because the probability of arriving in 25 minutes is more

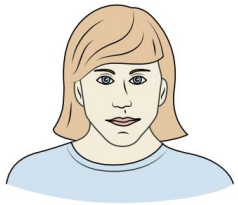
[2] Probabilistic Deep Learning: With Python, Keras and TensorFlow Probability, 2020

Why we need modeling uncertainty?

non-probabilistic classification vs **probabilistic classification** vs **bayesian probabilistic classification**

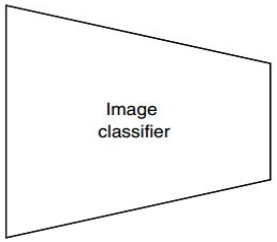
Problem: Face recognition

Person 1

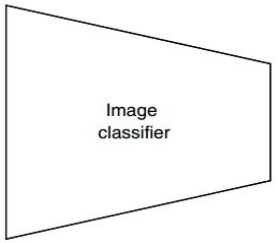


Person 2

1- **non-probabilistic classification** predicts just label.



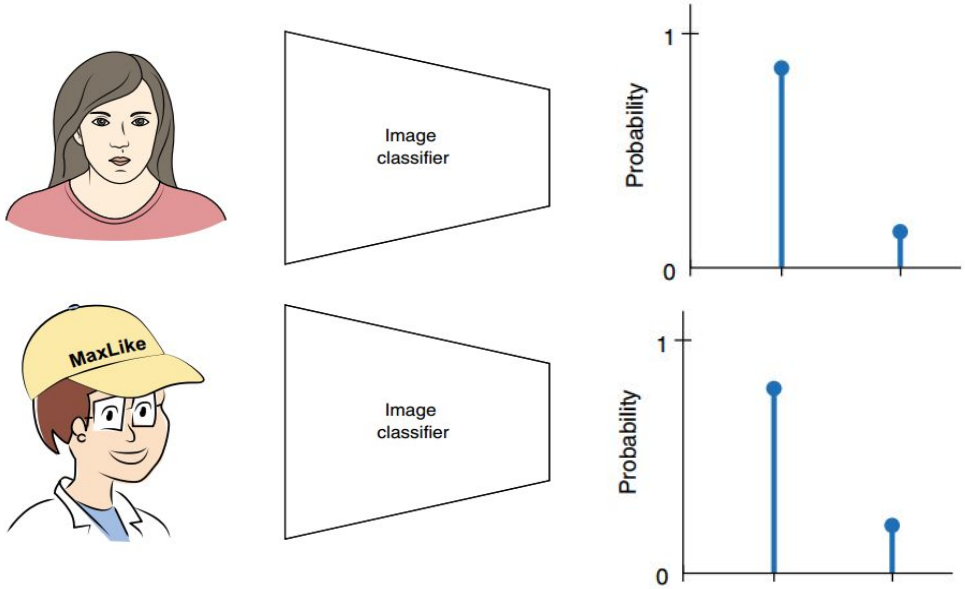
Person 1



Person 1

Why we need modeling uncertainty?

2- **probabilistic classification** predicts the labels with a probability assigned showing output uncertainty

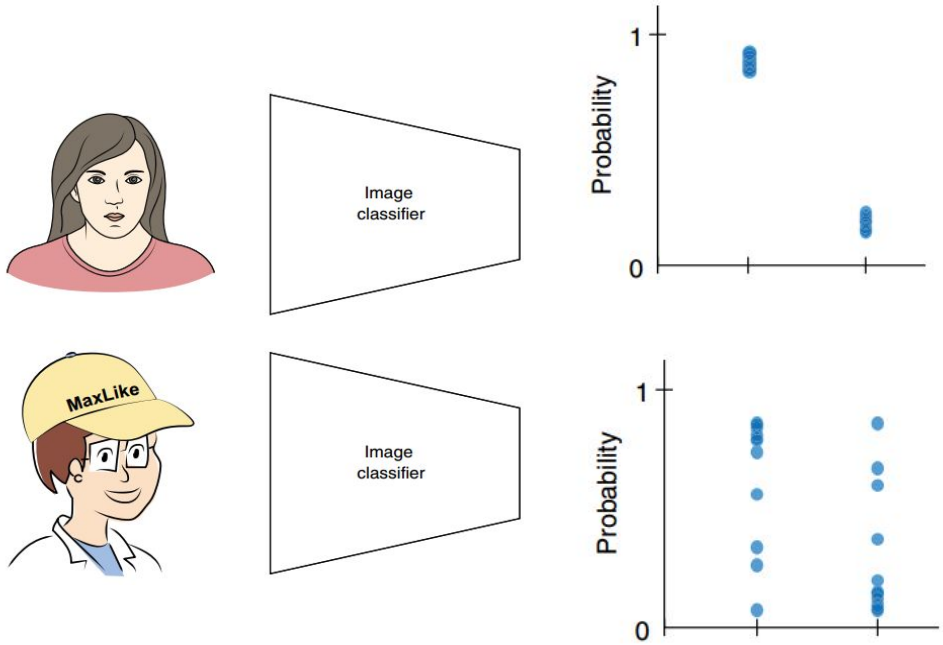


Why we need modeling uncertainty?

3- bayesian probabilistic classification

- How certain is the model about the assigned probability?
- Ask bayesian model several times and get different answers.

<u>This workshop is about the 3rd bayesian approach to model problems.</u>



What is bayesian thinking?

- You implement an algorithm.
- You are not certain that your code is bug-free.
- Your belief might be 50% sure your code is bug-free and 50% not.
- You start testing the code. After few tests you observed that tests passed correctly.
- So your belief changes to a 90% bug-free code.

- Congratulations! This is thinking bayesian.

Bayesian thinking = prior belief -> observing data -> posterior belief (with assigned probability)

Frequentist thinking = long-run frequency of an event

- As we gather N=infinity observations Bayesian thinking converges to frequentist thinking. So bayesian approach is suitable when we have little data because we are more uncertain

What is bayesian thinking?

Bayes theorem:

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

$$\propto P(X|A)P(A) \quad (\propto \text{ means "is proportional to"})$$

P(A): **prior** probability (our belief)

P(A|X): **posterior** probability after observing data *X*

P(X|A): **likelihood** probability (if our prior belief is correct what is the probability of observing data x)

likelihood: what is the probability of event X

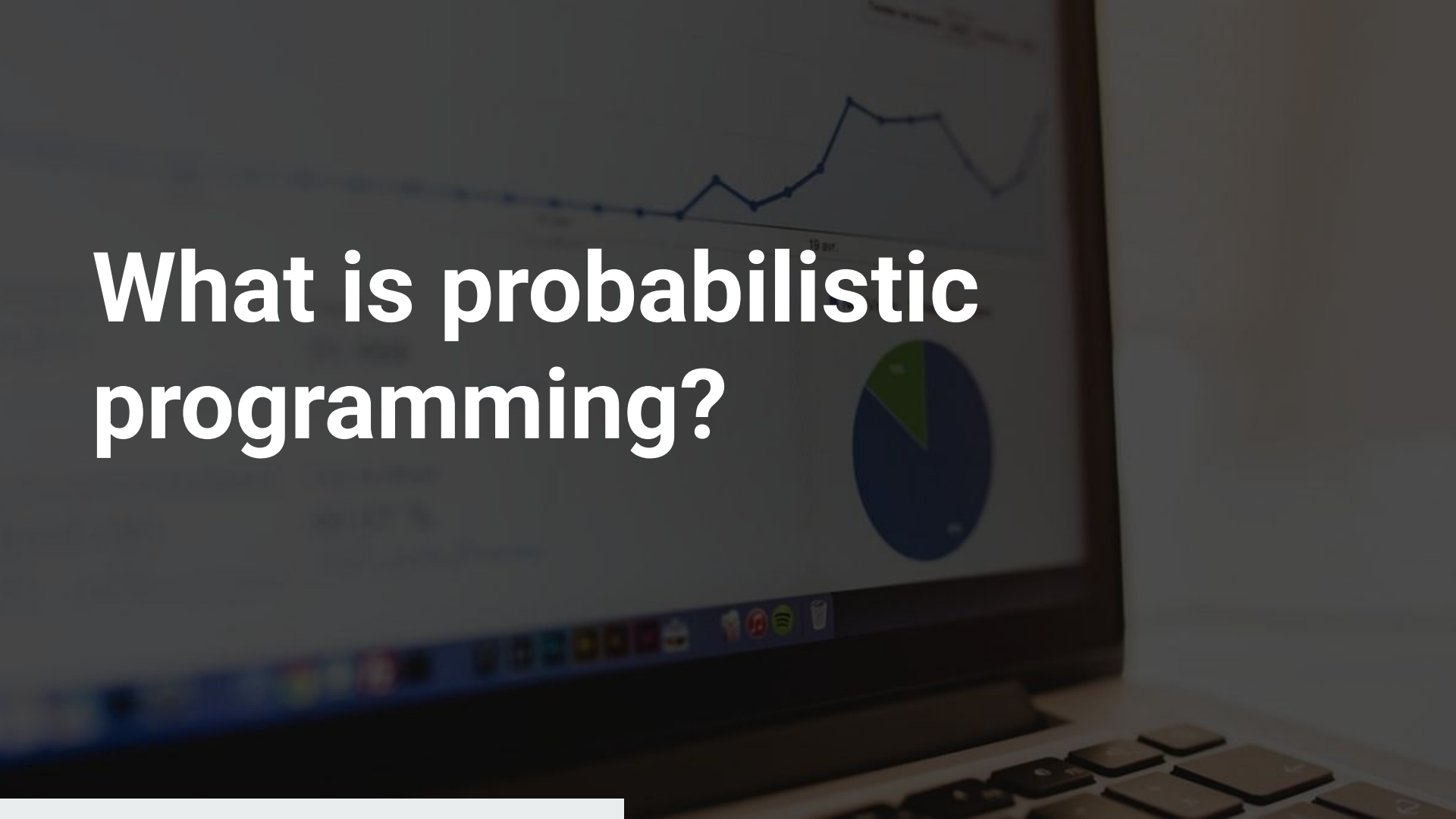
Maximum likelihood: For what parameters our probability of data X is maximum.

What is bayesian thinking?

Example (coding): **Coin flip**

- Our prior belief is we don’t know coin is fair.
- We begin to flip a coin and we update our belief.
- How our posterior belief changes after observing more and more heads or tails?

Let’s implement it! We will use **bernoulli, beta distribution**.



What is probabilistic programming?

Probabilistic Programming

We computed posterior probability of coin flipping after observing data.
Actually we find parameter of our posterior probability.

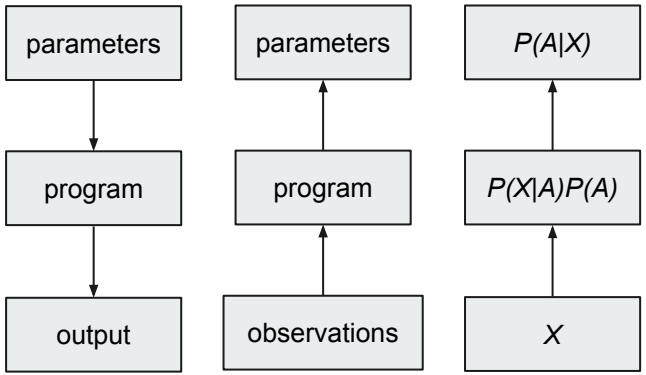
Probabilistic Programming is for estimating those parameters by just:

- Definition of probabilistic models
- Definition of input data
- And posterior probability automatically computed

Probabilistic Distributions Definition

Automatic parameter estimation

Probabilistic Programming (Example)




```
import tensorflow as tf
import tensorflow_probability as tfp

# Pretend to load synthetic data set.
features = tfp.distributions.Normal(loc=0.,
scale=1.).sample(int(100e3))

labels = tfp.distributions.Bernoulli(logits=1.618 *
features).sample()

# Specify model.
model = tfp.glm.Bernoulli()

# Fit model given data.
coeffs, linear_response, is_converged, num_iter = tfp.glm.fit(
    model_matrix=features[:, tf.newaxis],
    response=tf.cast(labels, dtype=tf.float32),
    model=model)
```

Probabilistic Graphical Models (PGMs).

Suppose we have a distribution **parameterized** by: $A_1, A_2, A_3, \ldots A_n$

- By chain rule we can write joint distribution(left of equation) equals to left conditional distributions

$$P(A_1,\ A_2,\ A_3,\ \ldots,\ A_n)\ =\ P(A_1).\,P(A_2 \mid A_1).\,P(A_3 \mid A_2,\ A_1)\ldots P(A_n \mid A_{n-1},\ \ldots,\ A_2,\ A_1)$$

Each conditional distribution might not depend on all its previous variables.

So we can each conditional depends on some other variables calls **parent**.

$$P(A_1,\ A_2,\ A_3,\ \ldots,\ A_n)\ =\ P(A_1 \mid Pa(A_1)).\ P(A_2 \mid Pa(A_2))\ldots P(A_n \mid Pa(A_n))$$

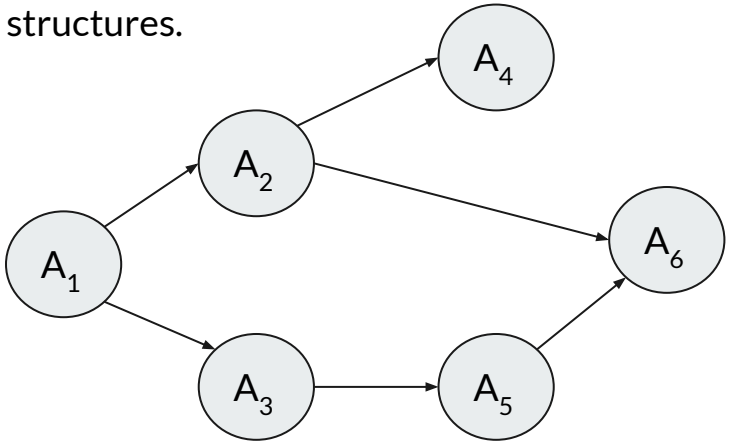
Probabilistic Graphical Models (PGMs).

PGMs are combination of probabilistic models with graph structures.

They used to show conditional dependencies between Variables.

All PGMs courses explain two most important parts:

- Representation
- Inference

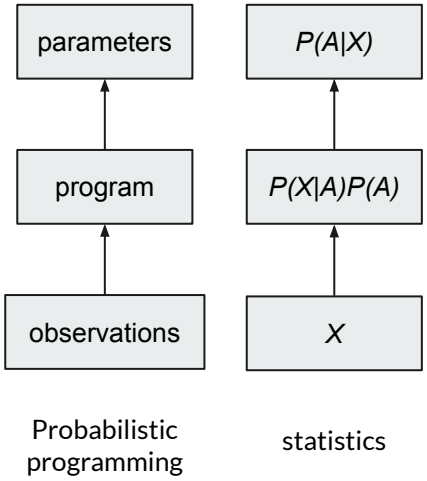


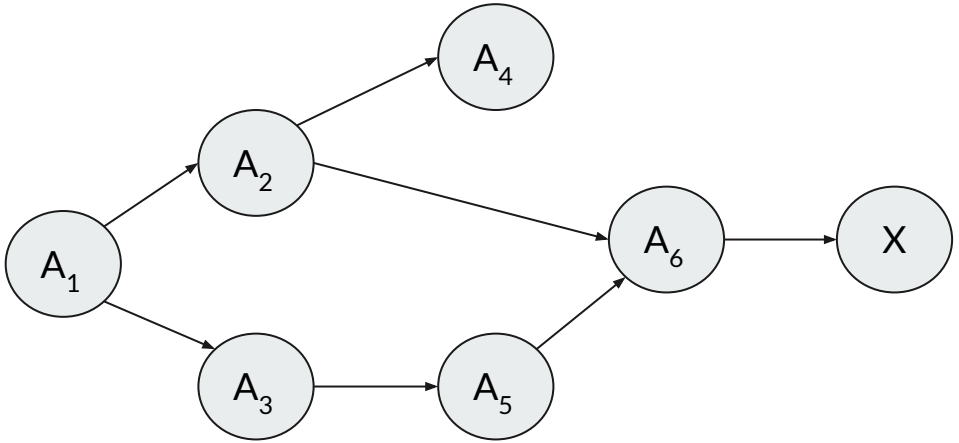
Representation and Inference used in probabilistic programming.

Let’s see what are these two based what we saw until now!

Representation.

Remember Probabilistic Program and Statistics:

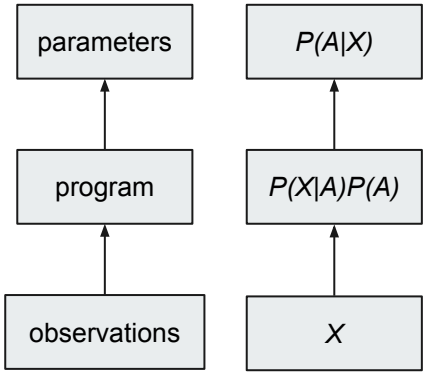


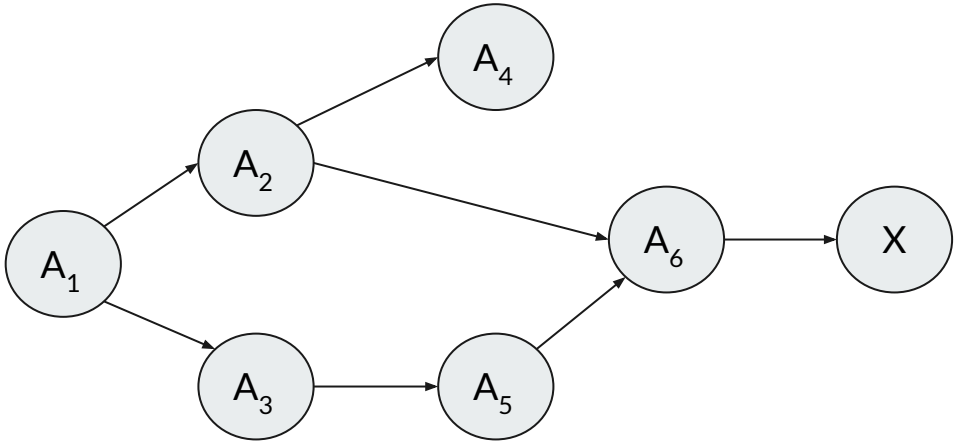


- We have some observation *X*
- We **<u>represent</u>** our program as a PGM that show how **parameters (A)** of our **probabilistic model** generate our data.

Inference

Remember Probabilistic Program and Statistics:



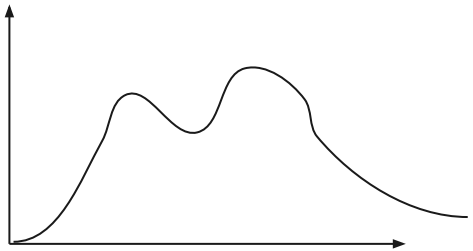


- we need to **estimate parameters(A)** of our models as posterior probability after observing data.
- Inference in general means ask questions from joint probability distribution of graph which is *P(A, X)*.

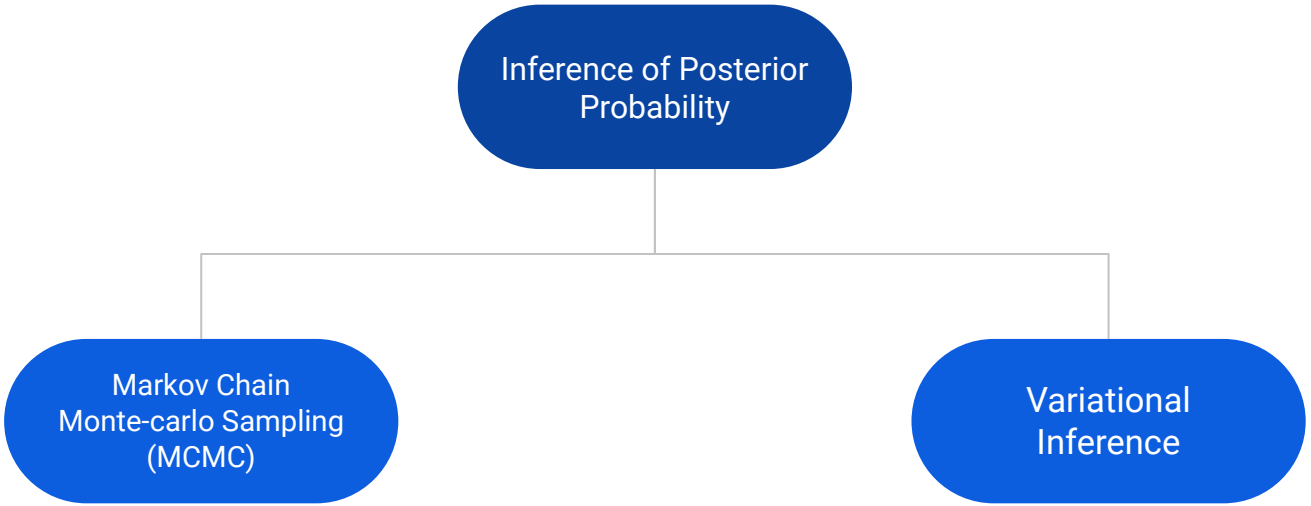
  For example as *P(A|X)* which is posterior probability

Posterior Probability (Inference)

- Computing Posterior Probability usually isn't as easy as coin flip example.
- There are two approach for this task which we will learn and use them in probabilistic programs in this workshop.



A



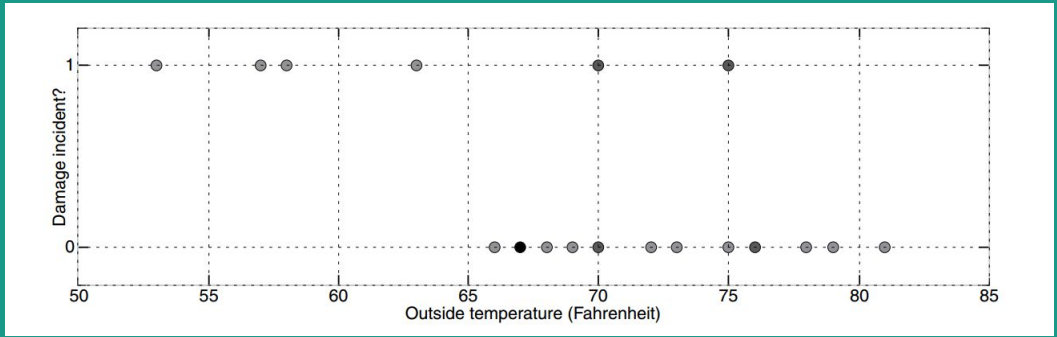
Our first Probabilistic Program

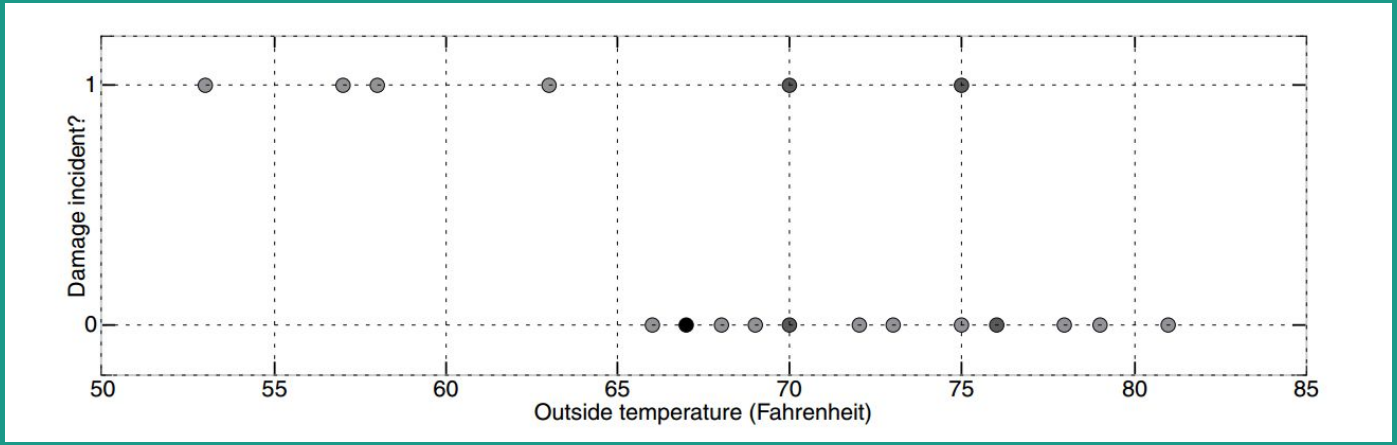
Challenger Space Shuttle Disaster.

Accident because of “O-Ring” failure.

Failure is because of outside temperature.

Before main flight, data of 23 test flights available

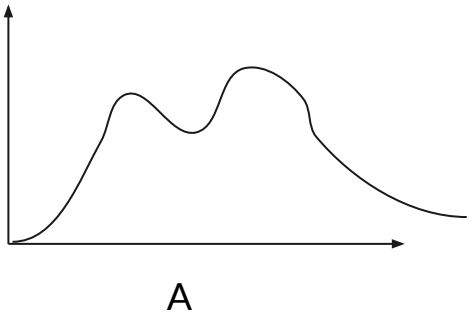




- We need model uncertainty because flight decision is risky.
- With tools we discussed before we need prediction uncertainty and parameter uncertainty.

Let's implement it by first designing our probabilistic program and finally compute posterior probability of parameters to reach desired output and parameters uncertainty. (PyMC3!)

Posterior Probability (Inference)



A

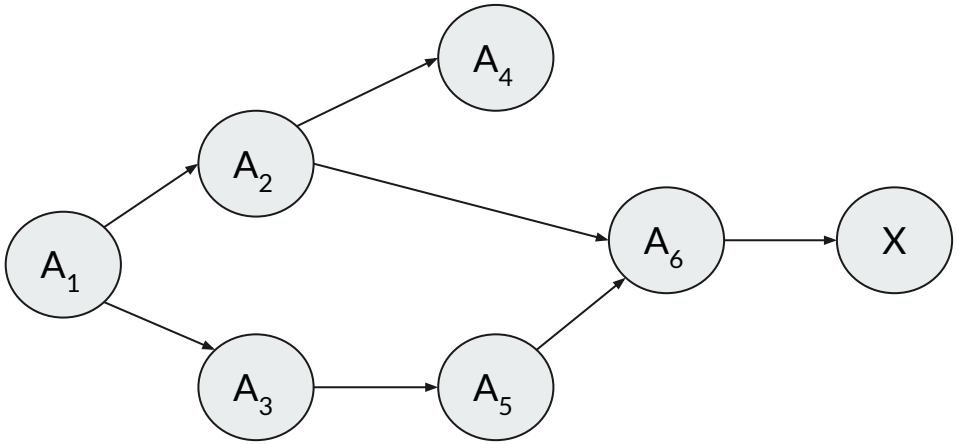
Inference of Posterior Probability

Markov Chain Monte-carlo Sampling (MCMC)

Variational Inference

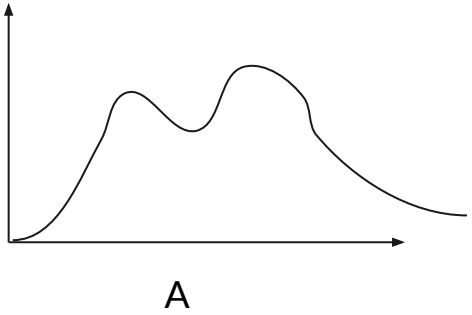
Markov Chain Monte-carlo Sampling (MCMC)

- We need to reach probability distribution of parameter *A* that we don't have any information from just knowing these parameters generate observation X.
- First Solution: Draw as many as independent random samples A and generate data to see which parameters are more probable.



This Called **Monte-Carlo** sampling.

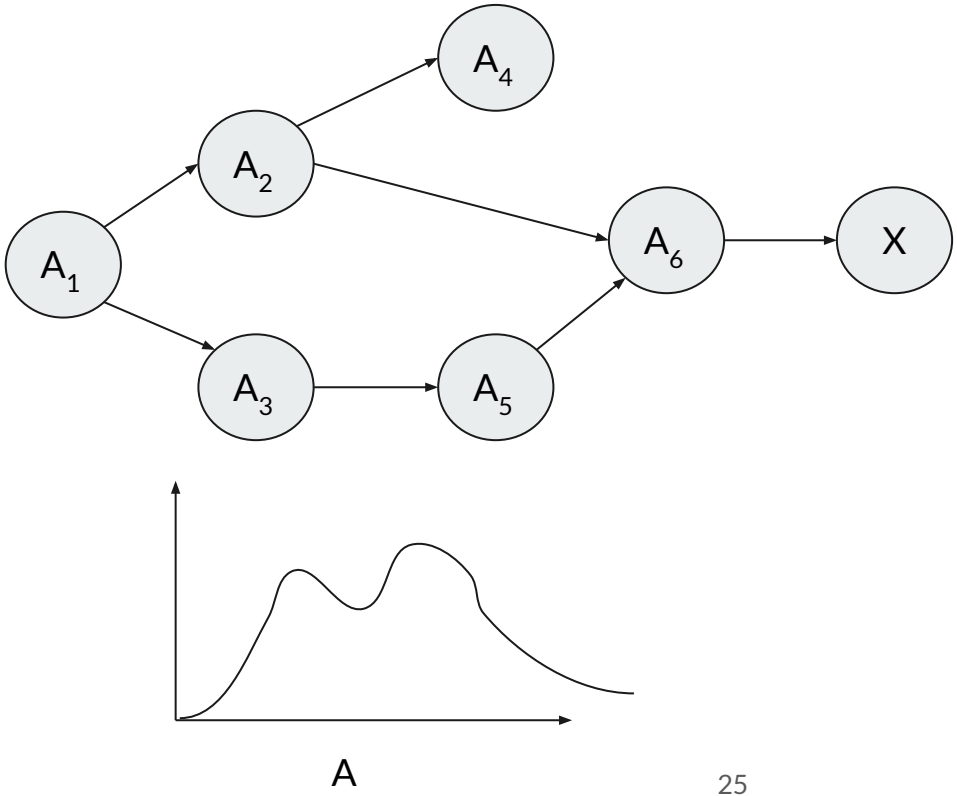
This simple idea is interactable for high dimension parameters space, because it has no intelligence.



Markov Chain Monte-carlo Sampling (MCMC)

Better solution: A strategy that starts from random sampling but draw dependent samples finally converges to stationary points where in samples are from the desired distribution.

This called markov chain because **samples are dependent** and called monte-carlo because still **we have sampling.**

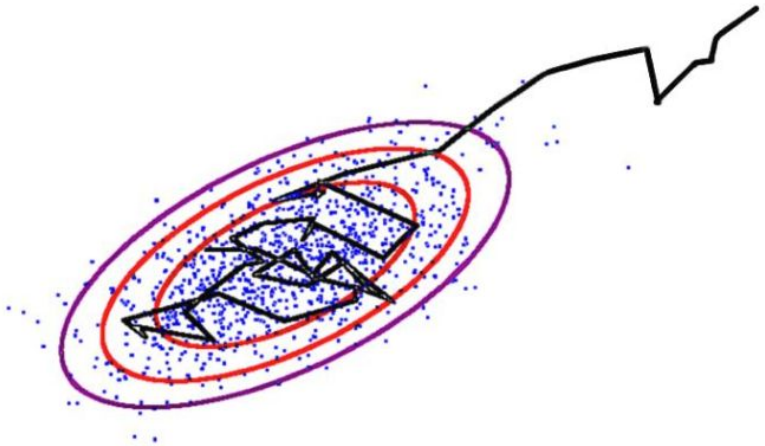


Markov Chain Monte-carlo Sampling (MCMC)

Metropolis algorithm (MCMC):

- Draw dependent samples from Q where Q is:

$$Q(A_{new} \mid A_{old}) = N(A_{new} \mid \mu = A_{old}, \sigma)$$



- Converge to the desired distribution by strategy P:
  If P is very low don't move to A new

$$P(A_{new} \mid A_{old}) = \min\left(1, \frac{P(A_{new})}{P(A_{old})}\right)$$

P(A) computed from PGM

Markov Chain Monte-carlo Sampling (MCMC)

PymC3 and Tensorflow Probability has more advanced MCMC methods that are faster and more intelligence (compute gradient of P(X) to move better directions):

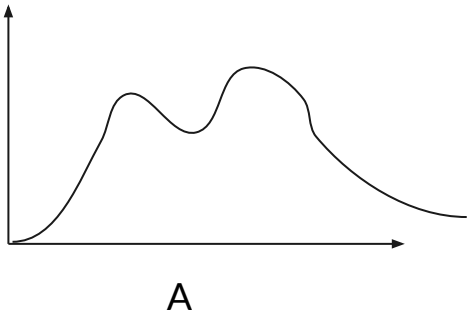
- No-U-Turn Sampler.
- Hamiltonian Metropolis

Although advanced methods of MCMC are faster and more suitable for high dimensional problems but generally MCMC methods have problems.

Problems of MCMC:

- Convergence in high dimensional data
- Slow when we have lots of data. (Also sampling is computationally expensive)

Posterior Probability (Inference)



A

Inference of Posterior Probability

Markov Chain Monte-carlo Sampling (MCMC)

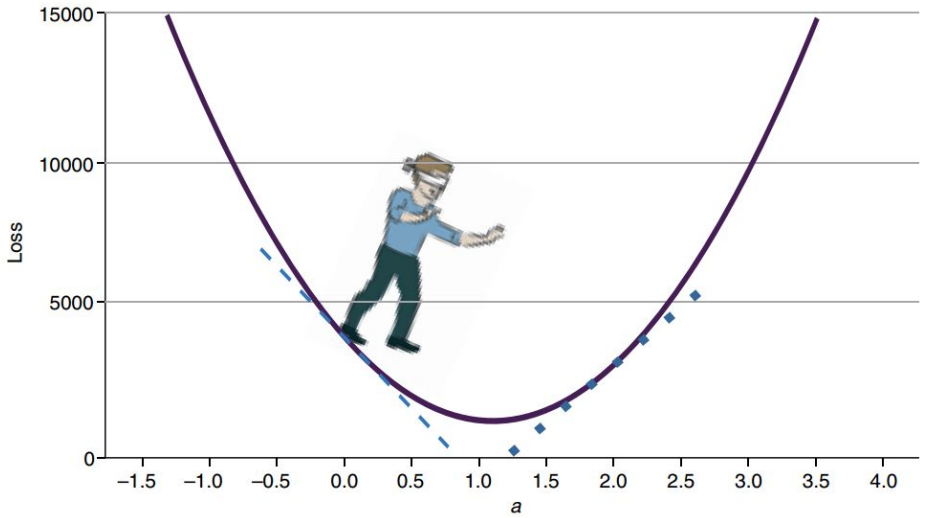
Variational Inference

Gradient Descent for optimization

optimization algorithm for finding a **local minimum** of a **differentiable** function

$$a_{t+1} = a_t - \eta \cdot \mathrm{grad}_a(\mathrm{loss})$$

- parameter **eta** control how much we go forward (learning rate).
- Modern deep learning frameworks like tensorflow and pytorch computes gradient automatically, called **Automatic Differentiation.**



KL divergence

KL is a measure of how one probability distribution is different from a second:

Distance between two Probability Distribution.

discrete distributions

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right).$$

continuous distributions

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

P=Q if and only if KL = 0

Variational Inference

Remember we need to compute posterior probability of parameters:

P(A|X)

As we saw computing this probability is interactable, because we have no information about A. One solution was MCMC but it has disadvantages.

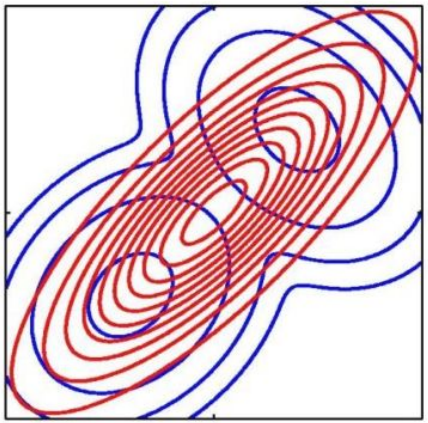
- Instead we approximate it with a proposal simpler distribution ***Q(A)*** (for example normal distribution)**.**
- Then we try to learn ***Q(A)*** as close as to ***P(A|X)***

- What is our distance metric for closeness? **KL-divergence**

Variational Inference

So we have an optimization problem!

$$Q^* = \underset{Q \in \mathcal{Q}}{\operatorname{argmin}}\, KL(P||Q)$$



Red is proposal distribution ***Q(A)*** (normal).
Blue is posterior distribution ***P(A|X)***.

Variational Inference

optimization problem: $$Q^* = \underset{Q \in \mathcal{Q}}{\operatorname{argmin}}\, KL(P||Q)$$

But wait! :(

There are **two problems**:

- We don't have ***P(A|X)***. How can we compute its kl-divergence?!
- But we have Our PGM and bayes theorem. We can compute approximation of ***P(A|X)***.

$$P(A|X) = \frac{P(X|A)P(A)}{P(X)}$$
$$\propto P(X|A)P(A) \quad (\propto \text{ means "is proportional to"})$$

Variational Inference

Because we change ***P(A|X)*** to ***P_hat(A|X)*** we missed some information.

So what happened.

If you continue to expand the kl divergence you reach below equation.
(https://arxiv.org/pdf/1907.08956.pdf)

$$Log\, P(x_i) \geq -D_{KL}(Q(A \mid x_i) \,||\, P(A)) + E_{Q(A \mid x_i)}[\log P(x_i \mid A)]$$

We can compute the right hand side of the equation which called **ELBO**.

It Becomes **Metric** for our optimization problem. We missed some information because **ELBO is lower bound for probability of data or *P(X)***. But still a reasonable metric.

Variational Inference

$$Log\, P(x_i) \geq -D_{KL}(Q(A \mid x_i) \,||\, P(A)) + E_{Q(A \mid x_i)}[\log P(x_i \mid A)]$$

- ***Q(A|x)*** is our proposal posterior distribution (normal)
- ***P(A)*** is our prior distribution for parameters.
- ***P(x|A)*** is ***P_hat(x|A)*** which computed from our PGM and data.
- Expectation ***E*** computed by **averaging!**

MCMC vs Variational Inference

- Variational inference is a lot faster than mcmc because it is an optimization problem, can be solved with gradient descent using gpu!

- MCMC methods a are guaranteed to find a globally optimal solution given enough time. (It's hard in practice) but:

- Variational inference finds locally optimal solution. Why?
  Because we consider a simpler distribution ***Q(A)*** for example normal to become as close as to ***P(A|X)***.

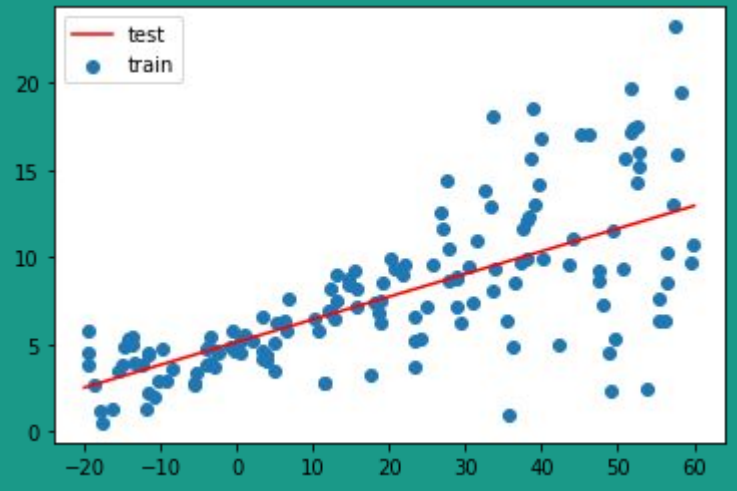
  But Don't Worry:

- There are solutions to enrich ***Q(A)*** with **normalizing flow** to get more complex distribution without missing anything!

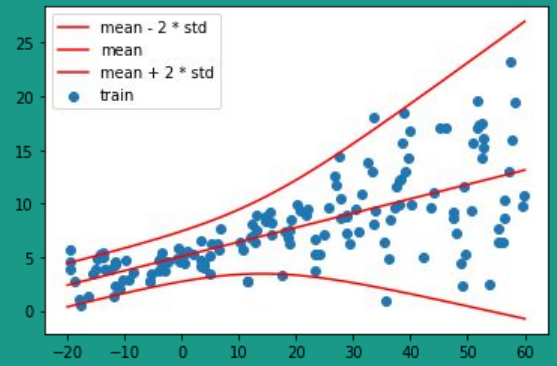
Bayesian Linear Regression (Implementation with Tensorflow Probability)

Task:

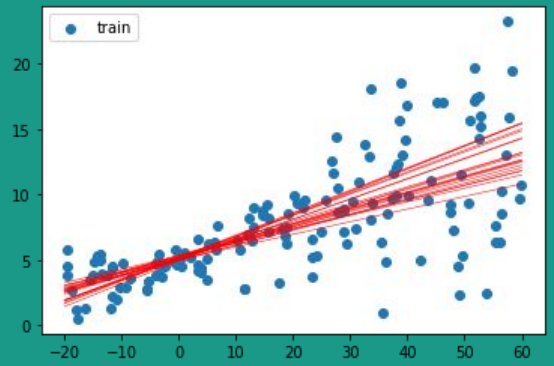
Find the line “y = ax + b”.
Actually find a, b



Output variability



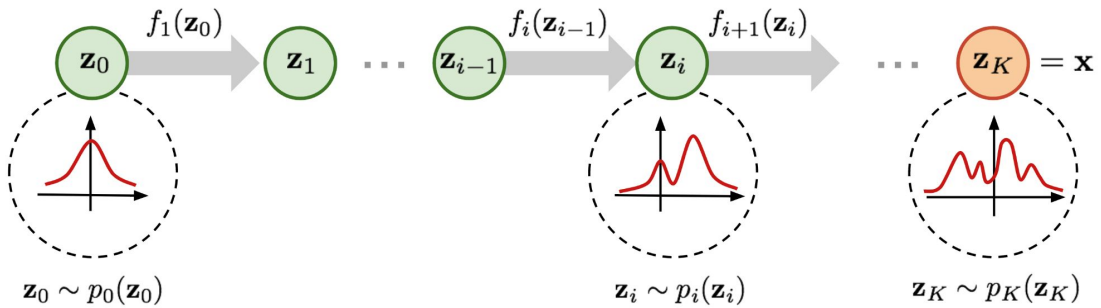
Parameter variability



Normalizing Flow

Tensorflow Bijectors: They can be used to transform distributions, preserving the ability to take samples and compute log_probs.

Normalizing Flow are transformations(bijectors) that convert simpler distributions to more complex ones.



Normalizing Flow

Change of Variables: Z and X be random variables which are related by a mapping $f : \mathbb{R}^n \to \mathbb{R}^n$ such that $X = f(Z)$ and $Z = f^{-1}(X)$. Then

$$p_X(\mathrm{x}) = p_Z(f^{-1}(\mathrm{x})) \left| \det \left(\frac{\partial f^{-1}(\mathrm{x})}{\partial \mathrm{x}} \right) \right|$$

1. The input and output dimensions must be the same.
2. The transformation must be invertible.
3. Computing the determinant of the Jacobian needs to be efficient (and differentiable)

Conclusion

We use Probabilistic programming when we need:

- **Parameter uncertainty**
- **Output Uncertainty**
- **The ability to inject domain knowledge into models**
- **Provide a framework for making risky decisions.**

Other libraries

https://docs.pymc.io/ (**PyMC3**): probabilistic programming in python.

https://www.tensorflow.org/probability (**TensorFlow Probability**): a library for probabilistic reasoning and statistical analysis.

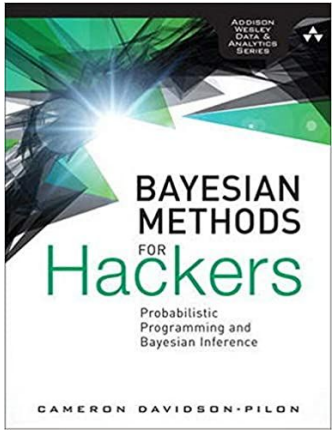
http://pyro.ai/ (**Pyro**): Deep Universal Probabilistic Programming.

https://probflow.readthedocs.io/en/latest/ (**ProbFlow**): A Python package for building Bayesian models with TensorFlow or PyTorch. Better API than tensorflow probability.

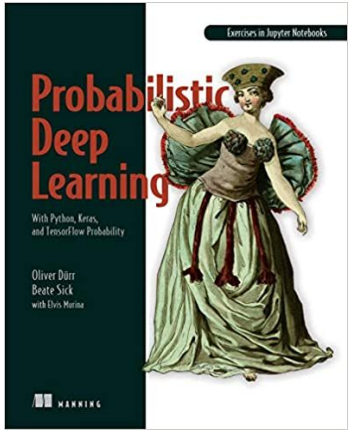
...

Books and Courses.

[1] ***Bayesian methods for hackers: probabilistic programming and Bayesian inference, 2015 ~ 250 pages.***



[2] ***Probabilistic Deep Learning: With Python, Keras and TensorFlow Probability, 2020 ~ 300 pages.***



[3] **Probabilistic Deep Learning with TensorFlow 2**
https://www.coursera.org/learn/probabilistic-deep-learning-with-tensorflow2

[4] CS 228 (stanford) - **Probabilistic Graphical Models**
Course notes: https://ermongroup.github.io/cs228-notes/

Main References

[1] Davidson-Pilon, C., 2015. ***Bayesian methods for hackers: probabilistic programming and Bayesian inference*.** Addison-Wesley Professional.

[2] Duerr, O., Sick, B. and Murina, E., 2020. ***Probabilistic Deep Learning: With Python, Keras and TensorFlow Probability*.** Manning Publications.

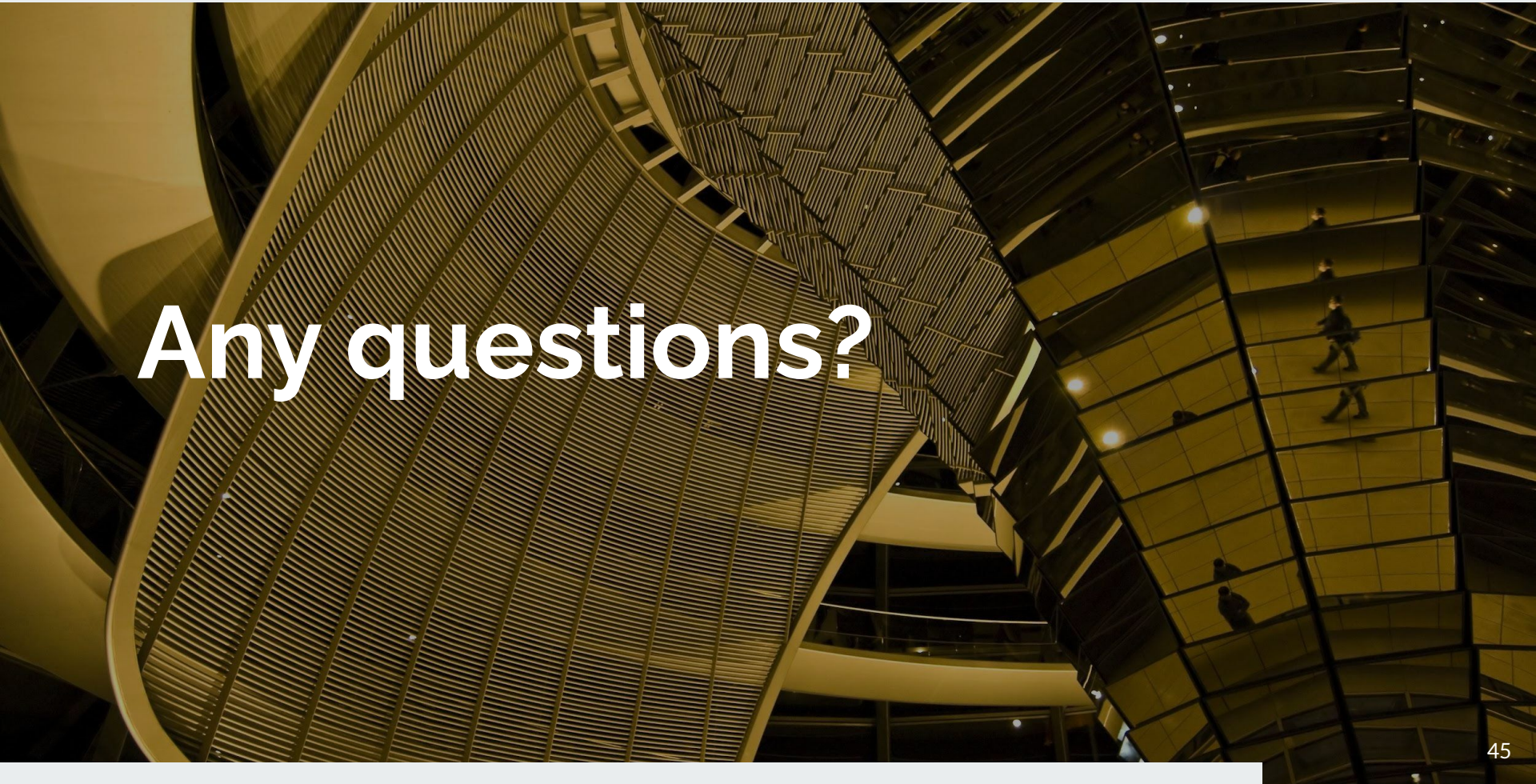
[3] Dillon, J.V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M. and Saurous, R.A., 2017. **Tensorflow distributions.** *arXiv preprint arXiv:1711.10604*.

[4] Salvatier, J., Wiecki, T.V. and Fonnesbeck, C., 2016. **Probabilistic programming in Python using PyMC3**. *PeerJ Computer Science*, *2*, p.e55.

[5] https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers
[6] https://www.tensorflow.org/probability
[7] https://docs.pymc.io/
[8] https://ermongroup.github.io/cs228-notes/

Other References

[1] https://www.youtube.com/watch?v=BrwKURU-wpk
[2] https://docs.scipy.org/doc/scipy/reference/stats.html
[3]https://fa.wikipedia.org/wiki/%D8%A7%D9%86%D9%81%D8%AC%D8%A7%D8%B1_%D9%81%D8%B6%D8%A7%D9%BE%DB%8C%D9%85%D8%A7%DB%8C_%DA%86%D9%84%D9%86%D8%AC%D8%B1
[4] https://blog.tensorflow.org/2018/12/an-introduction-to-probabilistic.html
[5] https://docs.pymc.io/pymc-examples/examples/getting_started.html
[6] https://docs.pymc.io/pymc-examples/examples/variational_inference/bayesian_neural_network_advi.html
[7] https://probflow.readthedocs.io/en/latest/user_guide/inference.html
[8] http://pyro.ai/examples/intro_part_ii.html
[9] https://ermongroup.github.io/cs228-notes/
[10] https://alexioannides.com/2018/11/07/bayesian-regression-in-pymc3-using-mcmc-variational-inference/
[11] https://arxiv.org/pdf/1601.00670.pdf
[12]https://colab.research.google.com/github/tensorflow/probability/blob/main/tensorflow_probability/examples/jupyter_notebooks/Variational_Inference_and_Joint_Distributions.ipynb
[13] https://blog.tensorflow.org/2019/03/regression-with-probabilistic-layers-in.html
[14] https://www.tensorflow.org/probability/api_docs/python/tfp/layers/DenseVariational
[15]https://arxiv.org/pdf/1907.08956.pdf


Any questions?

Thank you.

aliizadi2030@gmail.com