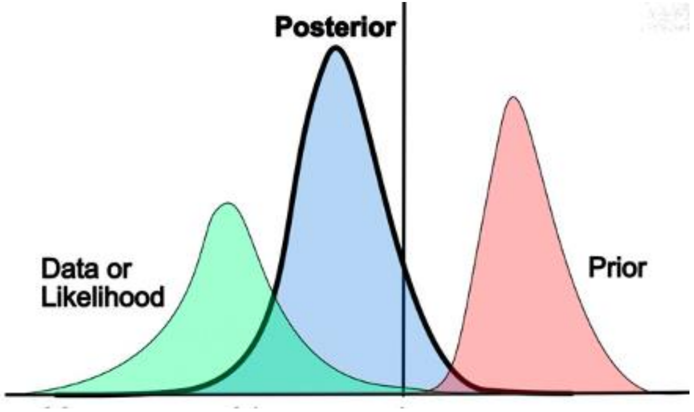
INFERENCES

Yuxiang.Qin@anu.edu.au
Woolley Building W22



Australian
National
University

W1-6

- Bash, shell, ssh
- High-Performance Computing
- Python
- C
- Parallel Computing

W7,8,9

- Data Processing
- Statistics
- Plotting
- Regression
- Clustering
- Model Selection

W10

- Neural Network

W11

- Simulations

W12

- Inference

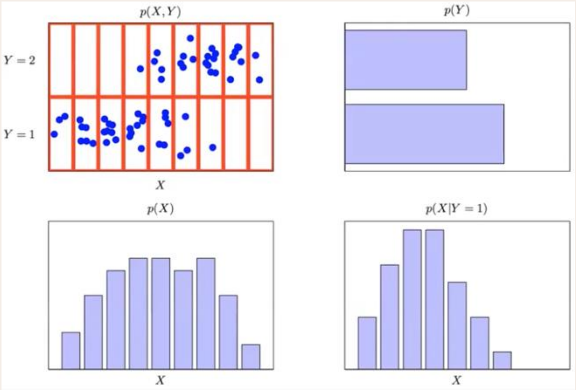
Inferences

using logic and reasoning to draw conclusions or make predictions about unknown information based on available data, evidence, or observations.

Deduction	Induction	Abduction
A conclusion that logically follows from a set of premises, with absolute certainty.	A conclusion based on specific observations, with some degree of uncertainty.	A conclusion based on the best explanation for a set of observations, with some degree of uncertainty.
All men are mortal. Socrates is a man. Therefore, Socrates is mortal. – Aristotle	A turkey found that every day for the past year he was fed at 9am sharp. Therefore, he concludes that "I am always fed at 9am". However, on the morning of Christmas eve, he had his throat cut. – Bertrand Russell	I had grasped the significance of the silence of the dog, for one true inference invariably suggests others.... Obviously, the midnight visitor was someone whom the dog knew well. – <The Adventure of Silver Blaze>
If a star's mass is above 10 solar masses, it will end its life in a supernova explosion. R136a1 has a mass of 200 solar masses. Therefore, R136a1 will end its life in a supernova explosion.	Many observed exoplanets have been found to orbit close to their host stars. It's likely that the newly discovered exoplanet Kepler-452b also orbits close to its star.	We observe the circular velocity of stars and gas in many galaxies remains flat in the outskirt of galaxies. Dark matter!

Bayesian Inference

updating our understanding based on new evidence



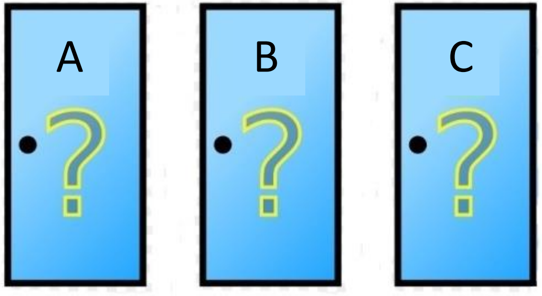
Recap W7

- Joint probability $\mathrm{P}(\mathrm{X},\mathrm{Y})$: the probability of X and Y
- Marginal probability $\mathrm{P}(\mathrm{X})$ or $\mathrm{P}(\mathrm{Y})$: the probability of X or Y
- Conditional probability $\mathrm{P}(\mathrm{X}|\mathrm{Y})$ or $\mathrm{P}(\mathrm{Y}|\mathrm{X})$: the probability of X given Y or of Y given X
- Sum rule, $\mathrm{P}(\mathrm{X}) = \sum_Y \mathrm{P}(\mathrm{X},\mathrm{Y})$, $\mathrm{P}(\mathrm{Y}) = \sum_X \mathrm{P}(\mathrm{X},\mathrm{Y})$
- Product rule, $\mathrm{P}(\mathrm{X},\mathrm{Y}) = \mathrm{P}(\mathrm{X}|\mathrm{Y})\mathrm{P}(\mathrm{Y}) = \mathrm{P}(\mathrm{Y}|\mathrm{X})\mathrm{P}(\mathrm{X})$: the probability of X and Y equals to the probability of X (or Y) times the probability of Y given X (or of X given Y)
- Bayes' rule: $\mathrm{P}(\mathrm{X}|\mathrm{Y}) = \mathrm{P}(\mathrm{Y}|\mathrm{X})\mathrm{P}(\mathrm{X})/\mathrm{P}(\mathrm{Y})$

X: the person having COVID; Y: the person tested positive on a RAT
Say 1% of people in this population are expected to have COVID, 90% of people with COVID will test positive on a RAT, 5% of all people tested positive (regardless of whether they have COVID or not)

i.e., P(having COVID) = 0.01, P(tested positive | having COVID) = 0.9, P(tested positive) = 0.05
P(having COVID | tested positive) 0.01 * 0.9 / 0.05 = 0.18

The Monty Hall problem



$$P(A|C) = P(C|A)P(A)/P(C)$$

P(A): the probability of the prize behind Door A (1/3) -> Prior

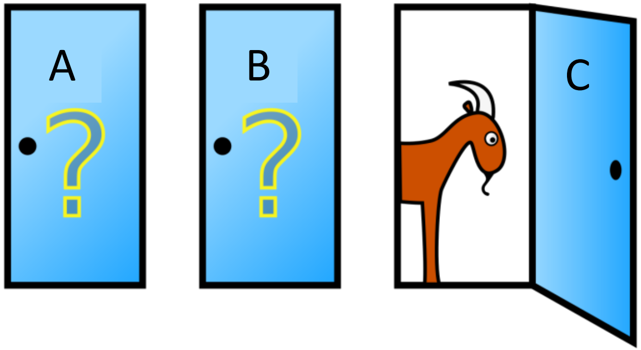
P(C|A): If the prize is behind Door A, the probability of the host opening Door C is 0.5

P(C): the probability of the host opening Door B:

- P(C |A) = 0.5
- P(C | not A):
  - P(C | If the prize is behind door B) = 1
  - P(C | If the prize is behind door C) = 0.

P(C) = (0.5+1+0) / 3 = 0.5

P(A|C) = P(C|A)P(A)/P(C) = 1/3 -> Posterior



Bayesian Inference

updating our understanding based on new evidence

X: your model parameters θ
Y: the observed data D

Say you look at the data points $(x_1, x_2, \ldots, x_N)$ and find that they seem to follow a normal distribution: $\theta \equiv (\mu, \sigma)$

$$\mathrm{P}(D|\theta) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{-\left(\frac{x_i-\mu}{\sqrt{2}\sigma}\right)^2}$$

Bayes' rule: $\mathrm{P}(\theta|D) = \mathrm{P}(D|\theta)\mathrm{P}(\theta)/\mathrm{P}(D)$

Recap W7

- Joint probability $\mathrm{P}(\mathrm{X}, \mathrm{Y})$: the probability of X and Y
- Marginal probability $\mathrm{P}(\mathrm{X})$ or $\mathrm{P}(\mathrm{Y})$: the probability of X or Y
- Conditional probability $\mathrm{P}(\mathrm{X}|\mathrm{Y})$ or $\mathrm{P}(\mathrm{Y}|\mathrm{X})$: the probability of X given Y or of Y given X
- Sum rule, $\mathrm{P}(\mathrm{X}) = \sum_Y \mathrm{P}(\mathrm{X}, \mathrm{Y})$, $\mathrm{P}(\mathrm{Y}) = \sum_X \mathrm{P}(\mathrm{X}, \mathrm{Y})$
- Product rule, $\mathrm{P}(\mathrm{X}, \mathrm{Y}) = \mathrm{P}(\mathrm{X}|\mathrm{Y})\mathrm{P}(\mathrm{Y}) = \mathrm{P}(\mathrm{Y}|\mathrm{X})\mathrm{P}(\mathrm{X})$: the probability of X and Y equals to the probability of X (or Y) times the probability of Y given X (or of X given Y)
- Bayes' rule: $\mathrm{P}(\mathrm{X}|\mathrm{Y}) = \mathrm{P}(\mathrm{Y}|\mathrm{X})\mathrm{P}(\mathrm{X})/\mathrm{P}(\mathrm{Y})$

- $\mathrm{P}(D|\theta)$: the likelihood function, representing the probability of observing the data (D) given a specific set of modeling parameters (θ)
- $\mathrm{P}(\theta)$: the prior probability distribution over the modeling parameters (θ). This represents our initial beliefs of possible values of θ
- $\mathrm{P}(D) \equiv \sum_\theta \mathrm{P}(D, \theta) = \sum_\theta \mathrm{P}(\theta)\mathrm{P}(D|\theta) = \int d\theta\, \mathrm{P}(\theta)\mathrm{P}(D|\theta)$: the marginal likelihood, representing the overall probability of observing the data (D)
- $\mathrm{P}(\theta|D)$: the posterior probability distribution over the modeling parameters (θ) given the data.

Bayesianism vs Frequentism

To a frequentist, probability does not represent degree of our belief.
Instead, it is only the frequency of a long-run experiment.

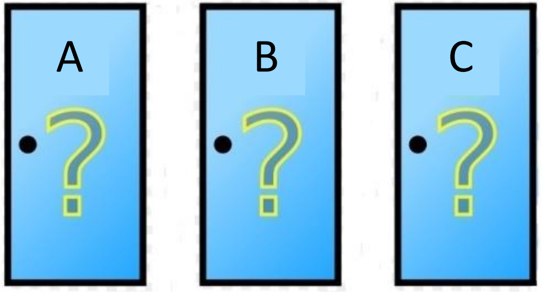
Example 1: When a frequentist flips a coin many many (infinitely many) times, half will be heads. But when a Bayesian statistician flips a coin many times and obtain half is head, they believe the probability for the next flip being head is half.

Example 2: When a frequentist is asked to determine the probability of flipping a coin to land on the head, they will keep flipping the coin many many times, collect data and calculate the frequency of head in the end of the experiment. But a Bayesian statistician will first guess e.g., 50% based on their prior knowledge of coin flipping and keep update the probability based on what they observe when flipping the coin

Example 3: When I flip one coin and hide the result, a Bayesian statistician will still say based on my prior knowledge, the probability of the coin being head is 50% and a frequentist will refuse to answer the question and say there is no probability because there is no long-run frequency, i.e., a probability.

Example 4: Pooling experts say there is a 51% probability this candidate will win the election while a frequentist will yell at them and say it makes no sense because there is only one election, so we cannot define a long-run frequency.

Monty Hall meets frequentists




```
def host_open_door(my_choice, win_door, door_options):
   # Determine which door the host opens
   for door in door_options:
      # cannot open my choice or the winning door
      if door != my_choice and door != win_door:
         return door


def my_switches(host_door, my_choice, door_options):
   # Simulate the contestant switching doors
   for door in door_options:
      # cannot be the original choice or the host door
      if door != my_choice and door != host_door :
         return door
```

```
door_options = ["A", "B", "C"]
win_by_switching = []
win_by_staying = []

# Simulate many games of Monty Hall
for game_num in range(10000):
   # Randomly choose the winning door
   win_door = random_choice(door_options)

   # Randomly choose a door for the contestant to initially choose
   my_choice = random_choice(door_options)

   # Host opens one of the other two doors that is not the winning one
   host_door = host_open_door(my_choice, win_door, door_options)

   # contestant switches doors if they want
   switch_door = switches(host_door, my_choice, door_options)

   if switch_door == win_door:
      win_by_switching.append(1)  # Switching won this game
      win_by_staying.append(0)  # Staying lost this game
   else:
      win_by_switching.append(0)  # Switching lost this game
      win_by_staying.append(1)  # Staying won this game

# Calculate the probabilities of winning
prob_win_by_switching = sum(win_by_switching)/10000
prob_win_by_staying   = sum(win_by_staying)/10000
```

Bayesian inference pseudocode

Say you look at the data points $(x_1, x_2, \ldots x_N)$ and find that they seem to follow a normal distribution: $\theta \equiv (\mu = 5, \sigma)$

```
# Initial prior and samples
prior = initial_prior
std_samples = sample_parameters(initial_prior, std_min, std_max, num_samples)
mean_guessed = 5

prior_samples      = np.zeros(num_samples)
likelihood_samples  = np.zeros(num_samples)
posterior_samples   = np.zeros(num_samples)

# Inference loop
for istep in range(num_steps):

    for i in range(num_samples):
        prior_samples[i] = prior(std_samples[i])
        likelihood_samples[i] = np.prod(norm_pdf(data, mean_guessed, std_samples[i]))
        posterior_samples[i] = likelihood_samples[i] * prior_samples[i]

    marginal_likelihoods = np.sum(likelihood_samples) / num_samples
    posterior_samples   /= marginal_likelihoods

    # Interpolate posterior for next step's prior
    f_interp = interp1d(std_samples, posterior_samples)
    def prior(mean, std):
        return f_interp(mean , std)

    std_samples = sample_parameters(prior, std_min, std_max, num_samples)
```

$\mathrm{P}(\theta|D) = \mathrm{P}(D|\theta)\mathrm{P}(\theta)/\mathrm{P}(D)$

$\mathrm{P}(D|\theta)$: likelihood

$\mathrm{P}(\theta)$: prior probability

$\mathrm{P}(D)$: marginal likelihood

$\mathrm{P}(\theta|D)$: posterior probability

Points in a sample might only have a uniform volume when initialized to be uniform and are very likely not the case at later times!

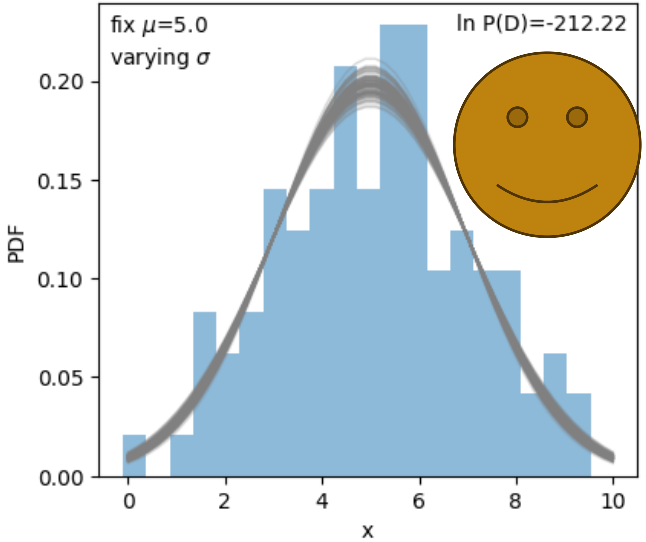
open inference-part1.ipynb

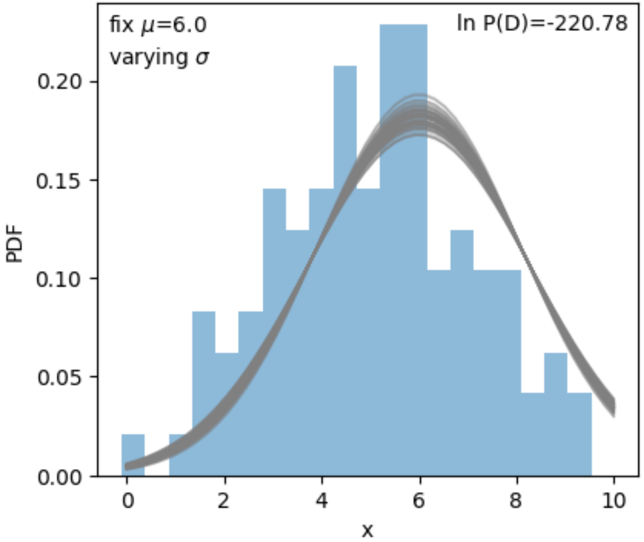
Bayesian evidence

$\mathrm{P}(D) \equiv \sum_\theta \mathrm{P}(D,\theta) = \sum_\theta \mathrm{P}(\theta)\mathrm{P}(D|\theta) = \int d\theta\, \mathrm{P}(\theta)\mathrm{P}(D|\theta)$:

The marginal likelihood, representing the overall probability of observing the data (D)

$$P(D|\mathcal{M}_1) = \int d\theta\, \mathrm{P}(\theta|\mathcal{M}_1)\mathrm{P}(D|\theta,\mathcal{M}_1) \qquad P(D|\mathcal{M}_2) = \int d\theta\, \mathrm{P}(\theta|\mathcal{M}_2)\mathrm{P}(D|\theta,\mathcal{M}_2)$$





A higher evidence indicates the model is more likely to have generated the observed data.

open inference-part2.ipynb

Bayesian evidence

Occam's razor: introducing additional parameters penalizes the evidence unless preferred by the data

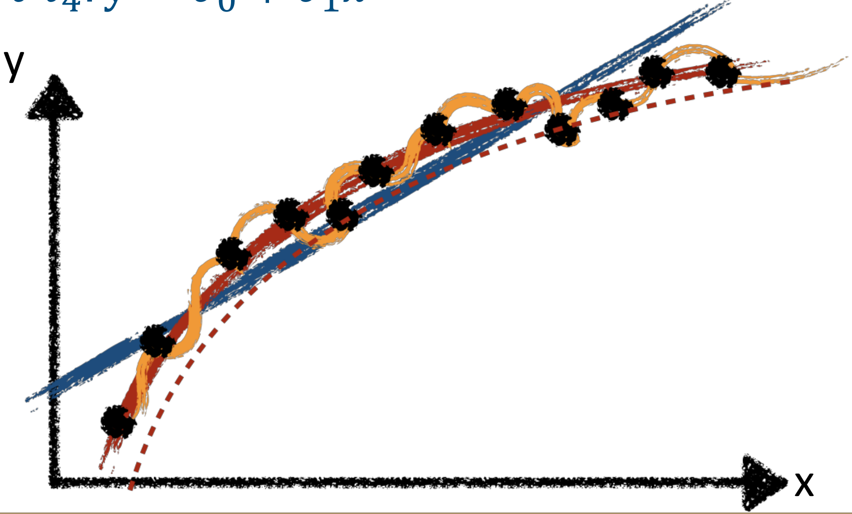
$\mathrm{P}(D) = \int d\theta\, \mathrm{P}(\theta)\mathrm{P}(D|\theta)$: the marginal likelihood, representing the overall probability of observing the data (D)

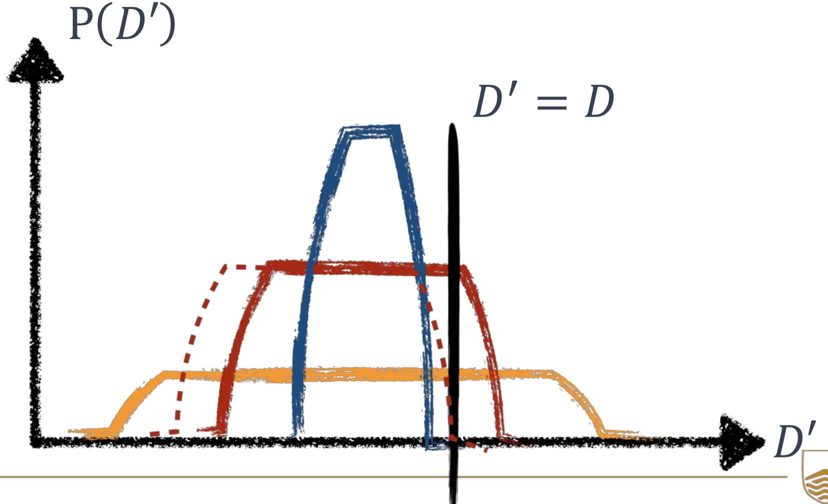
$\mathcal{M}_1\text{: y} = -1 + \theta_1 x + \theta_2 x^2$ — A good model with the right number of free parameters

$\mathcal{M}_2\text{: y} = -2 + \theta_1 x + \theta_2 x^2$ — A poor model with the same number of free parameters

$\mathcal{M}_3\text{: y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_n x^n$ — A very complex model that is able to predict different possible dataset but with low probability

$\mathcal{M}_4\text{: y} = \theta_0 + \theta_1 x$ — A model that is too simple to explain the observation





open inference-part3.ipynb

Nested sampling

Say you look at the data points $(x_1, x_2, \ldots x_N)$ and find that they seem to follow a normal distribution: $\theta \equiv (\mu = 5, \sigma)$

$$\mathrm{P}(\theta|D) = \mathrm{P}(D|\theta)\mathrm{P}(\theta)/\mathrm{P}(D)$$

$\mathrm{P}(D|\theta)$: likelihood
$\mathrm{P}(\theta)$: prior probability
$\mathrm{P}(D)$: marginal likelihood
$\mathrm{P}(\theta|D)$: posterior probability

```
# Initial prior and samples
prior = initial_prior
std_samples = sample_parameters(initial_prior, std_min, std_max, num_samples)
mean_guessed = 5

prior_samples      = np.zeros(num_samples)
likelihood_samples  = np.zeros(num_samples)
posterior_samples   = np.zeros(num_samples)

# Inference loop
for istep in range(num_steps):

  for i in range(num_samples):
    prior_samples[i] = prior(std_samples[i])
    likelihood_samples[i] = np.prod(norm_pdf(data, mean_guessed, std_samples[i]))
    posterior_samples[i] = likelihood_samples[i] * prior_samples[i]

  marginal_likelihoods = np.sum(likelihood_samples) / num_samples
  posterior_samples   /= marginal_likelihoods

  # Interpolate posterior for next step's prior
  f_interp = interp1d(std_samples, posterior_samples)
  def prior(mean, std):
    return f_interp(mean , std)

  std_samples = sample_parameters(prior, std_min, std_max, num_samples)
```

Points in a sample might only have a uniform volume when initialized to be uniform and are very likely not the case at later times!

Nested sampling

$$P(\theta|D) = P(D|\theta)P(\theta)/P(D)$$

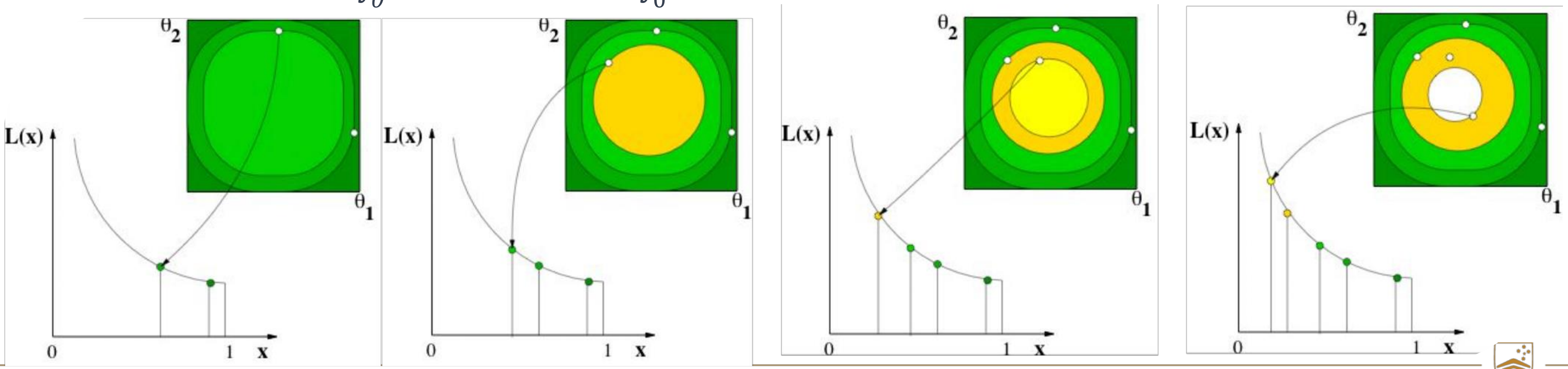
$P(D|\theta)$: likelihood
$P(\theta)$: prior probability
$P(D)$: marginal likelihood
$P(\theta|D)$: posterior probability

- We define iso-likelihood contour, which encapsulates/nests parameter space (θ) where likelihoods, $L(\theta) \equiv P(D|\theta)$, are higher than λ.
- We re-write our prior volume as:

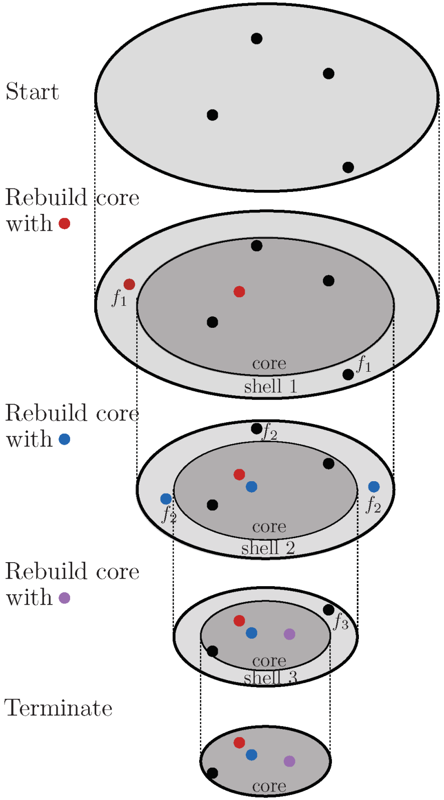
$$X(\lambda) = \int_{L(\theta)>\lambda} \mathrm{d}\theta P(\theta)$$

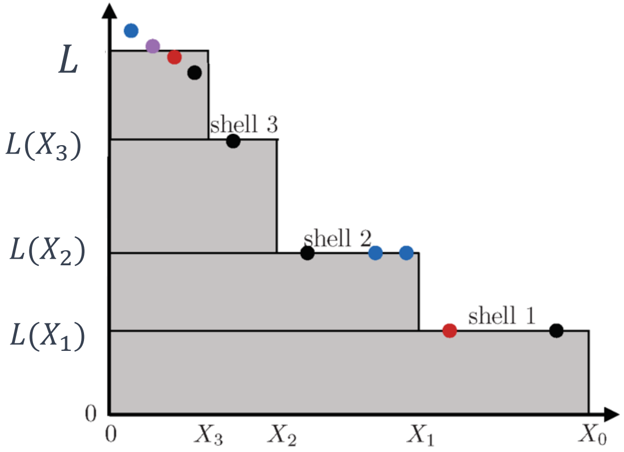
- We re-write the evidence in the space of prior volume, $X(\lambda)$:

$$Z = P(D) = \int_{\theta} \mathrm{d}\theta P(D|\theta)P(\theta) = \int_0^1 \mathrm{dX}\, L(X) = \sum L(X_i)\Delta X_i$$

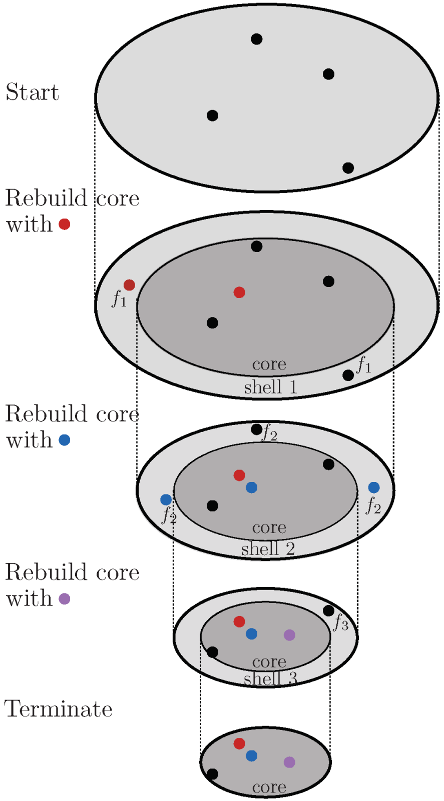


Nested sampling pseudocode



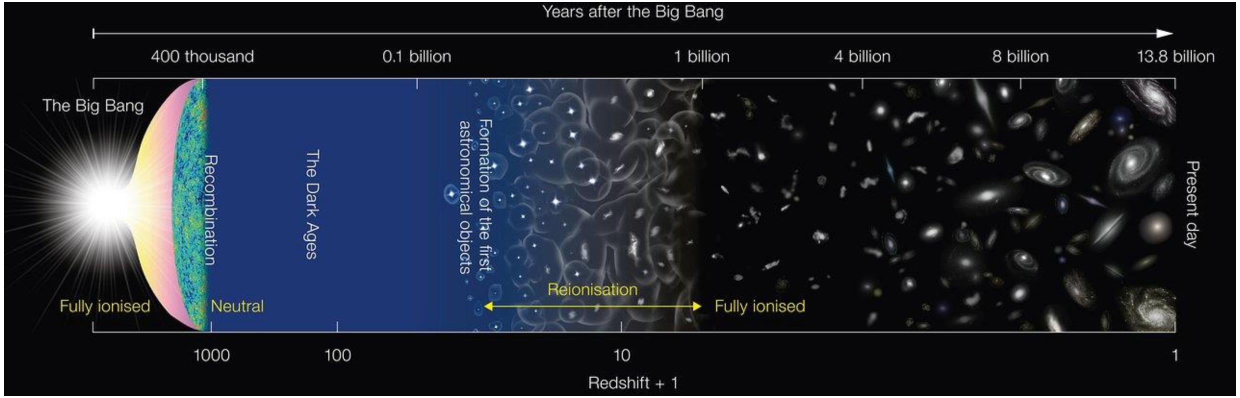


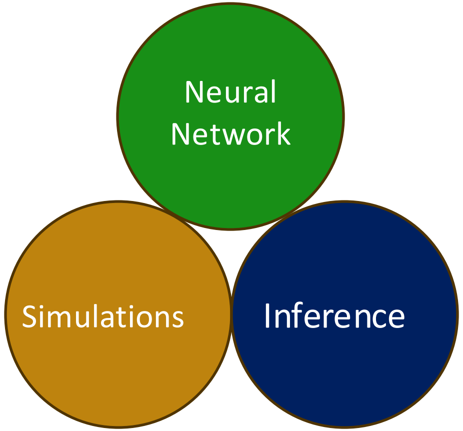
Nested sampling pseudocode

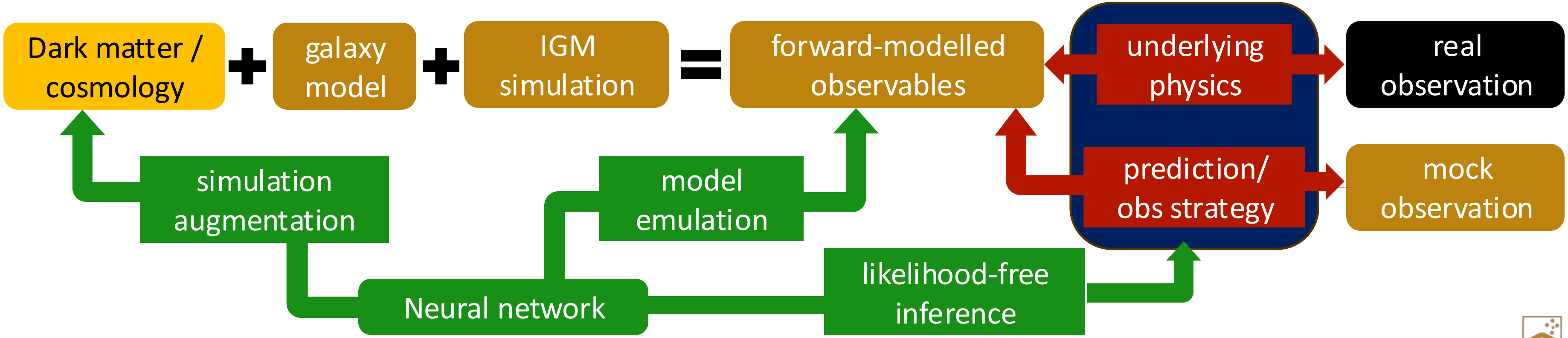


open inference-part4.ipynb

A bit about my research





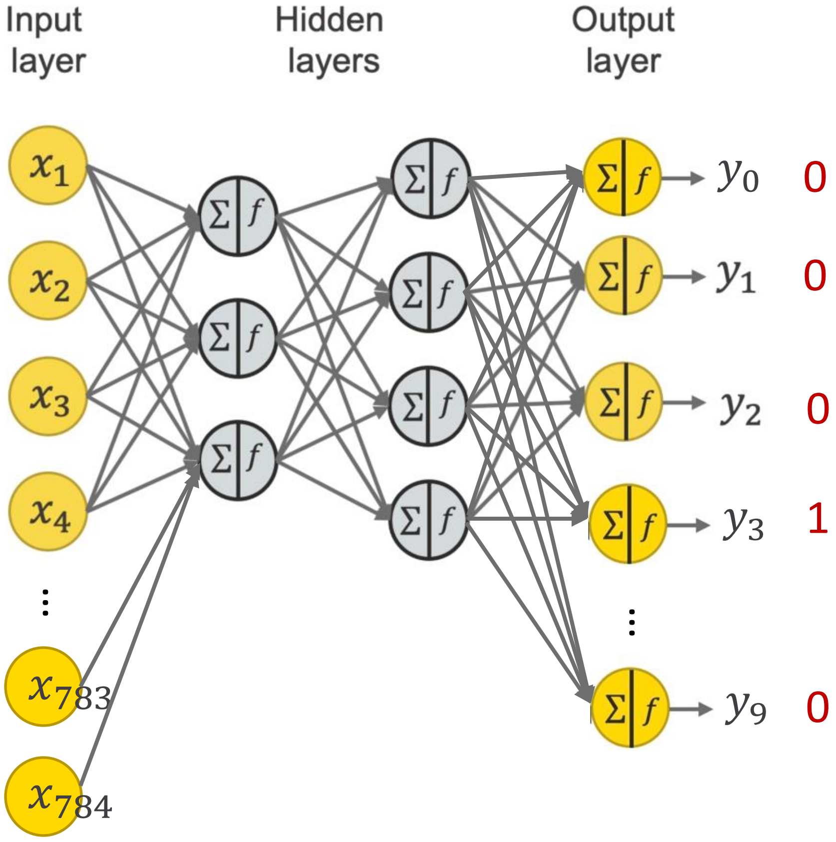


Recap W10

Artificial Neural Network

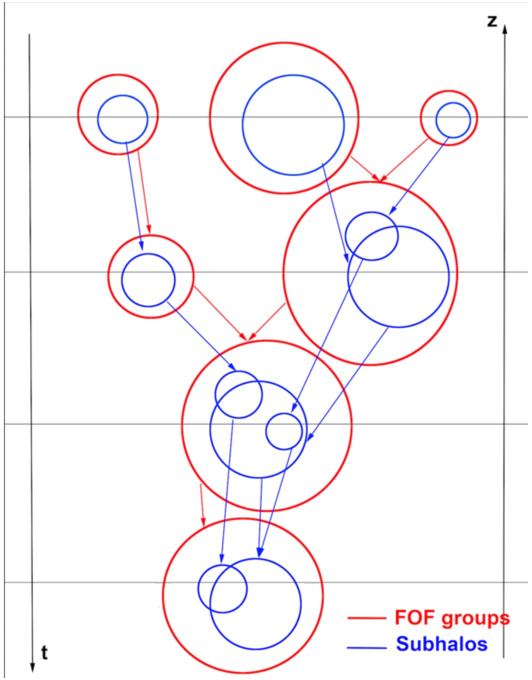
INPUT
(28x28)

INPUT
(784)

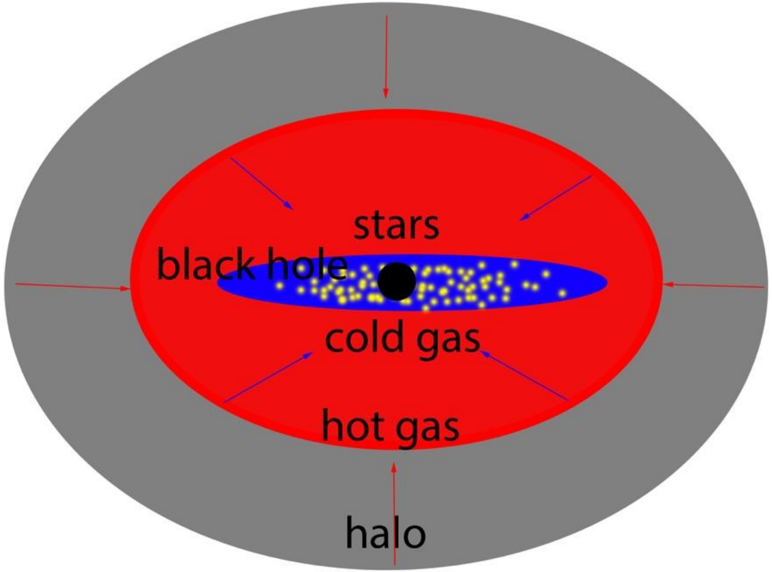


Semi-analytic model

Halo Merger Trees



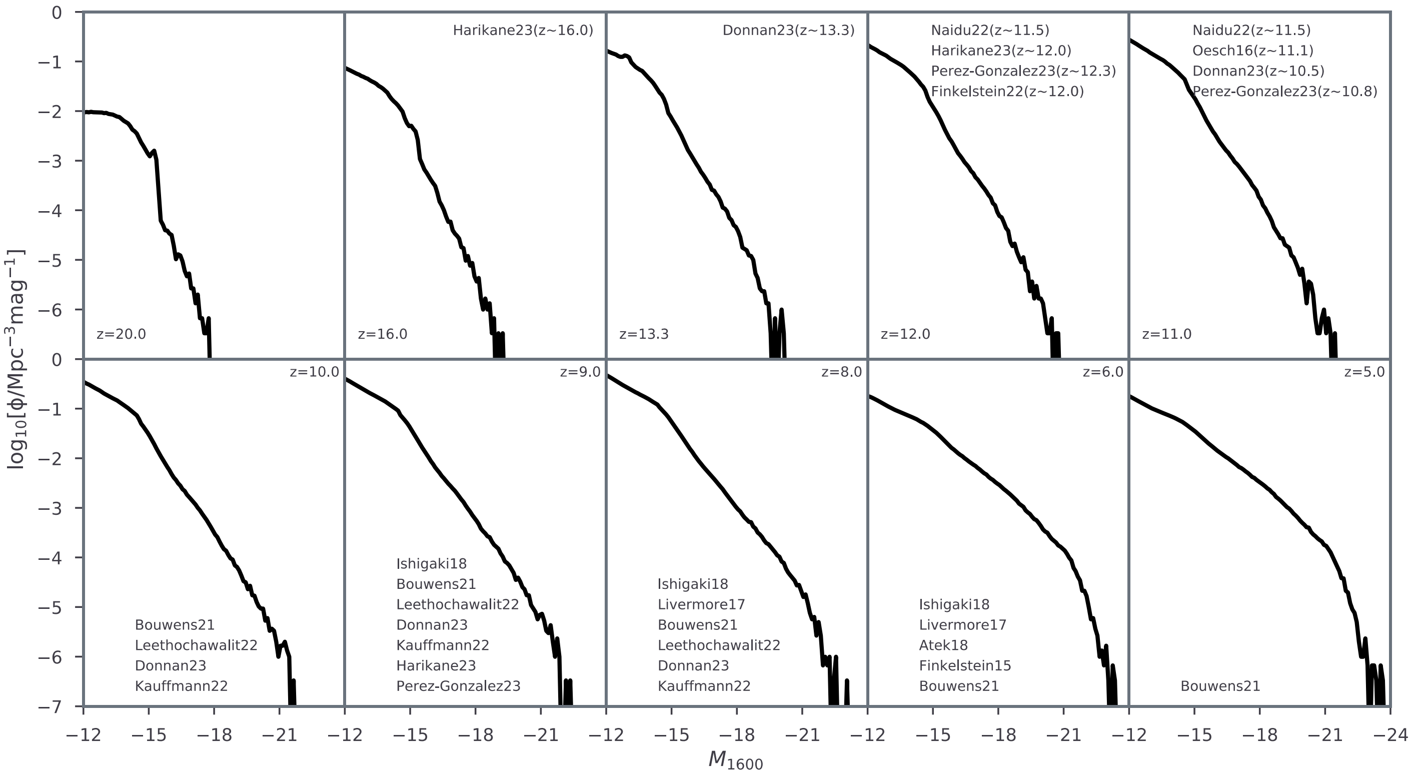
Semi-Analytic Model



Baryonic Physics

Gas infall
Gas cooling
Star Formation
Supernovae
Metal Enrichment
Black Hole Growth
Reionization Heating
AGN Feedback
Star Burst
...

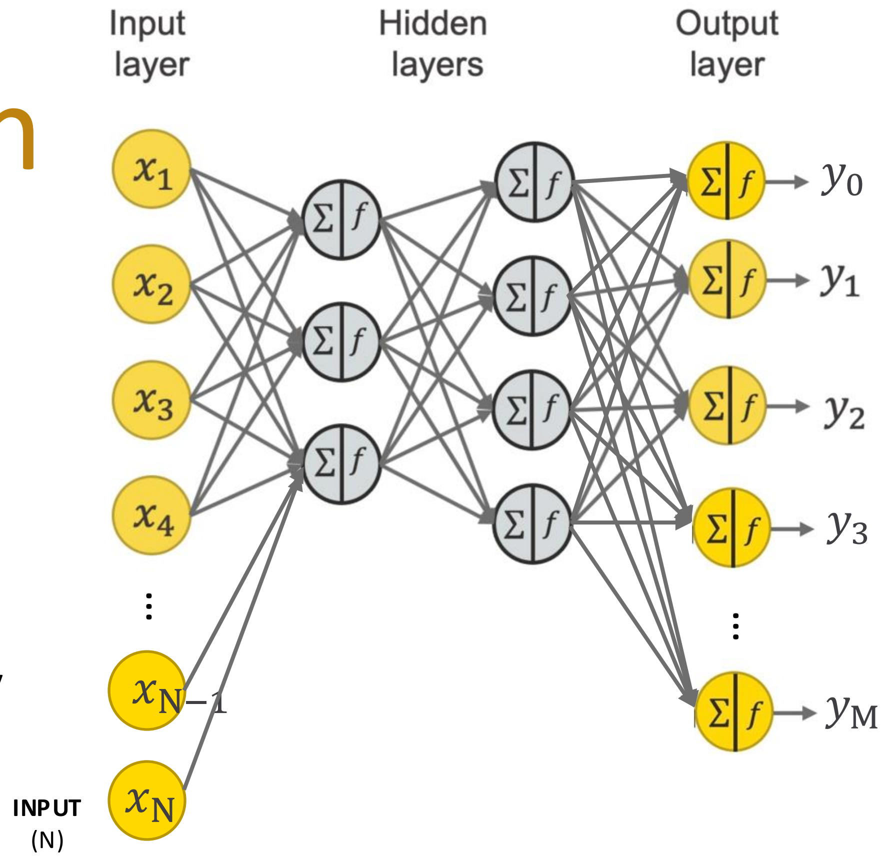
Population study using SAM

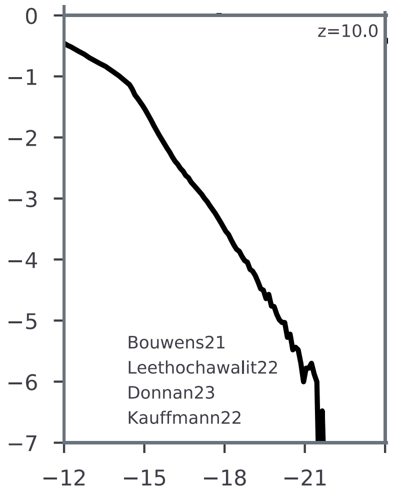


Model Emulation

Model parameters

Gas infall efficiency
Gas cooling efficiency
Star Formation efficiency
Supernovae efficiency
Metal Enrichment efficiency
Black Hole Growth efficiency
Reionization Heating efficiency
AGN Feedback efficiency
Star Burst efficiency
...





Likelihood-free inference
Simulation-based inference (SBI)

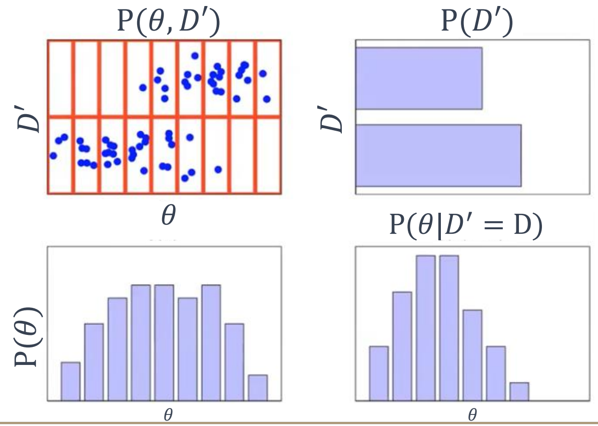
$$P(\theta|D) = P(D|\theta)P(\theta)/P(D)$$

$P(D|\theta)$: likelihood
$P(\theta)$: prior probability
$P(D)$: marginal likelihood
$P(\theta|D)$: posterior probability

What if we don't know the likelihood or there is no functional form for the likelihood?

Remember our model can reproduce different outputs (D') for given θ. This means that we can measure the frequency of those outputs that reproduce the real data (D).

Approximate Bayesian Computation




```
def approximate_bayesian_computation(observed_data, num_samples, tolerance):
    accepted_parameters = []

    for i in range(num_samples):
        # 1. Sample a parameter theta from the prior
        theta = sample_prior()

        # 2. Simulate data using the parameter theta
        simulated_data = simulate_data(theta)

        # 3. Compare simulated data to observed data
        distance = calculate_distance(simulated_data, observed_data)

        # 4. Accept or reject the parameter based on the distance
        if distance < tolerance: accepted_parameters.append(theta)

    return accepted_parameters
```

Likelihood-free inference

Simulation-based inference (SBI)

$\mathrm{P}(\theta|D) = \mathrm{P}(D|\theta)\mathrm{P}(\theta)/\mathrm{P}(D)$

$\mathrm{P}(D|\theta)$: likelihood

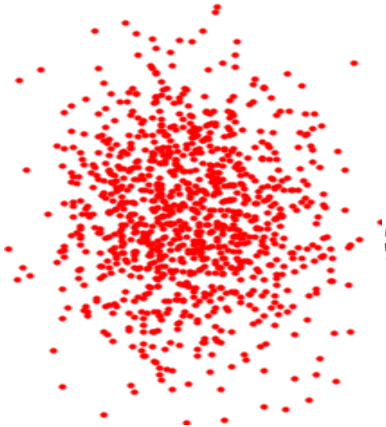
$\mathrm{P}(\theta)$: prior probability

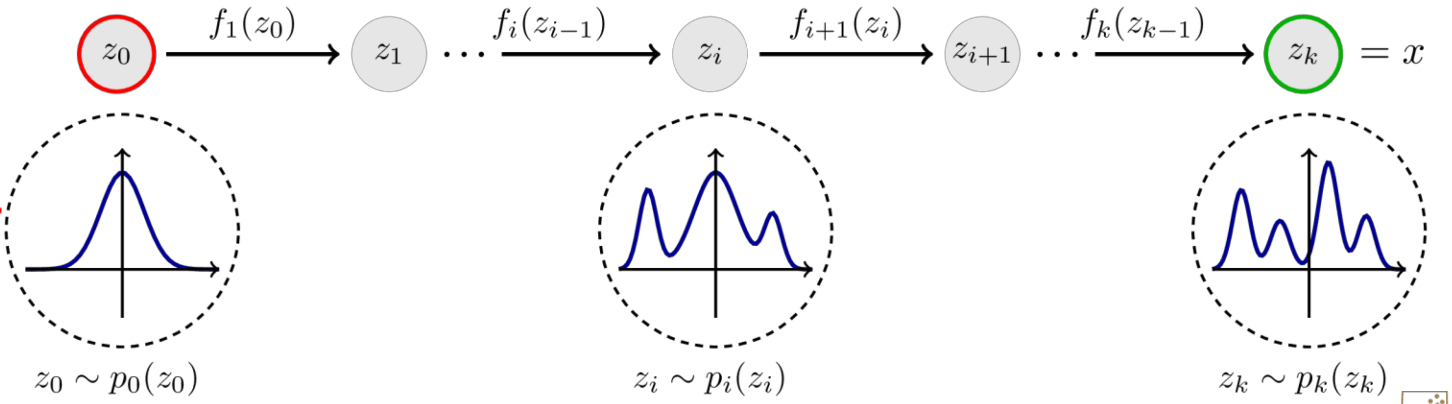
$\mathrm{P}(D)$: marginal likelihood

$\mathrm{P}(\theta|D)$: posterior probability

What if we can use a few sample of (θ, D') to estimate $\mathrm{P}(\theta|D)$?

Normalizing flow: A chain of invertible transformations constituting a normalizing flow, converting a simple distribution $p_0(z_0)$ into a complex one $p_k(z_k)$ step by step. The flows are trained to fit $p_k(z_k)$ to the target distribution $\mathrm{p}(x)$, e.g., $\mathrm{P}(\theta|D)$, $\mathrm{P}(D|\theta)$, $P(\mathrm{D}, \theta)$, ...



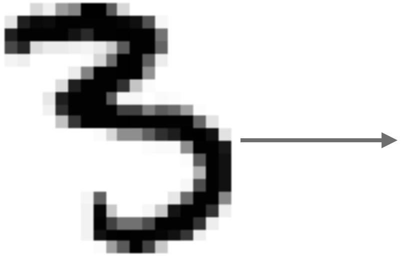


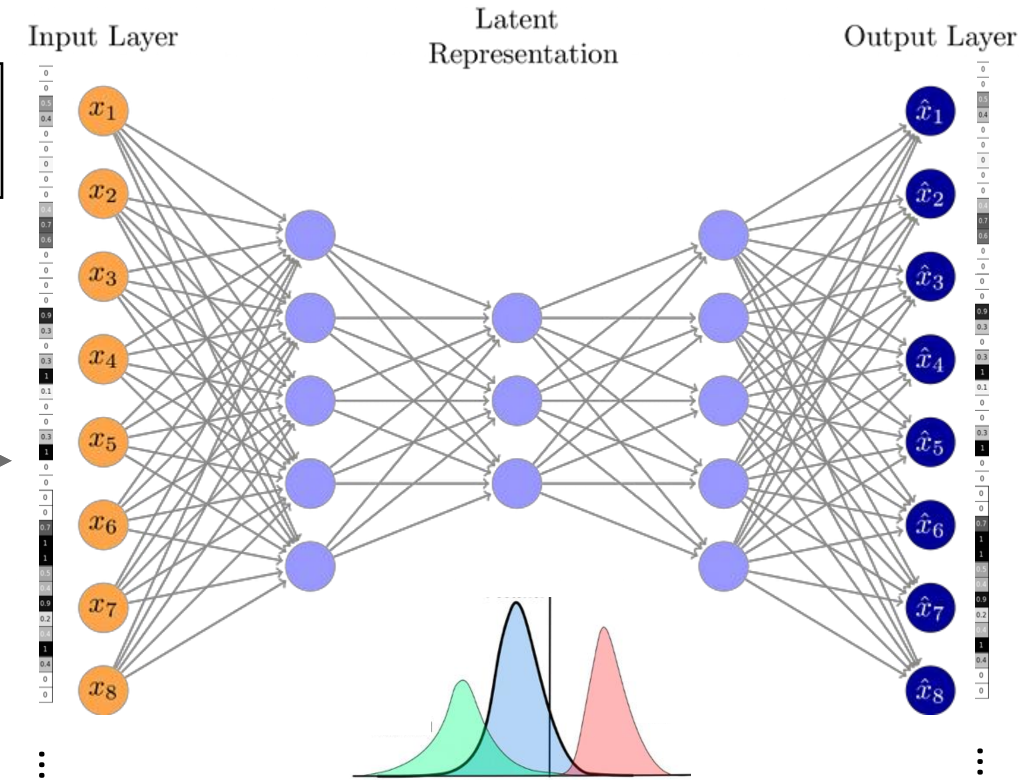
open inference-part5.ipynb

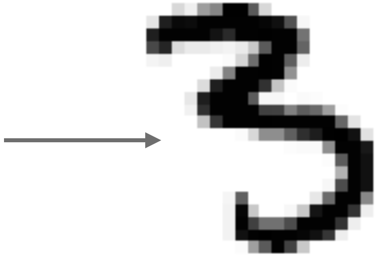
Generative models - Autoencoder

Encoder compresses the input data into a lower-dimensional latent space

Decoder reconstruct the original input data from the latent representation





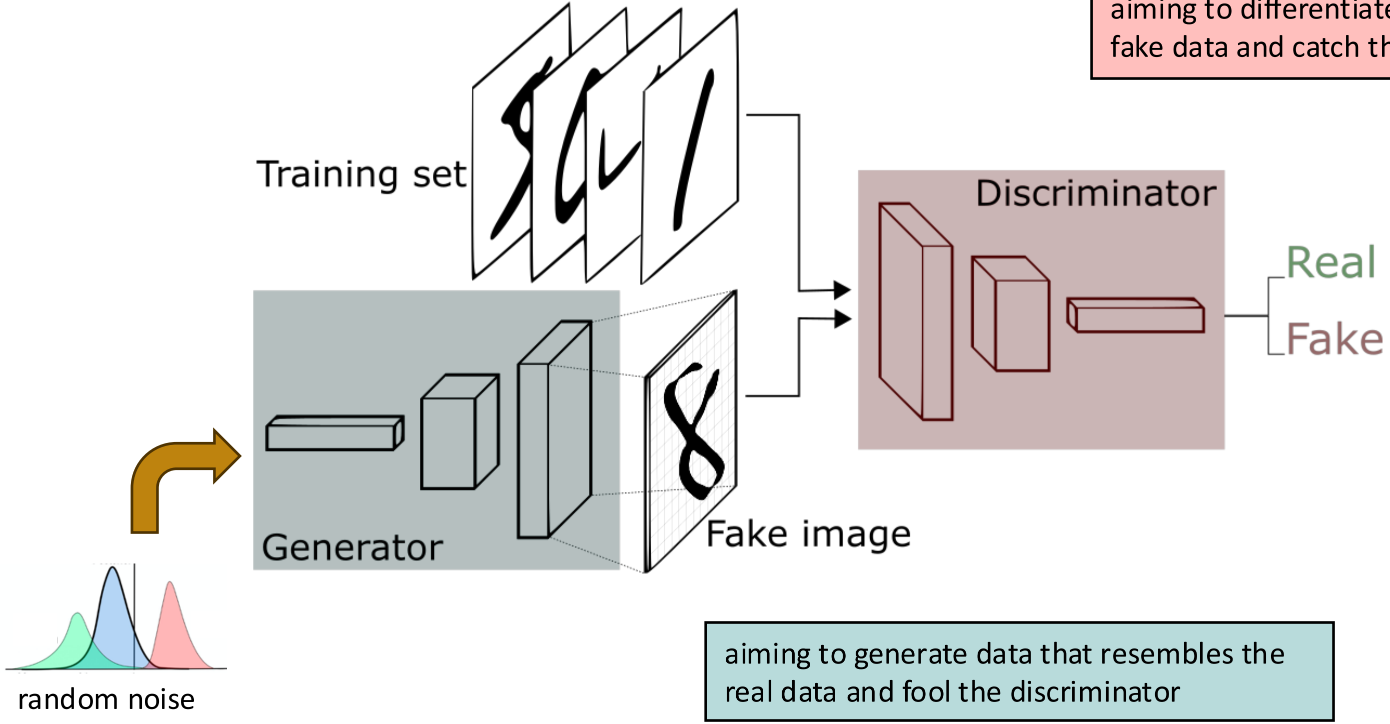


open neural-network-part4.ipynb

Generative models – GAN

(Generative Adversarial Network)

aiming to differentiate between real and fake data and catch the generator



aiming to generate data that resembles the real data and fool the discriminator

GAN pseudocode

```
generator = models.Sequential([
   layers.Input(shape=(z_dim,)), # Input latent variable
   ...
   layers.Dense(28x28,)]) # Output image
```

```
discriminator = models.Sequential([
   layers.Input(shape=(28x28,)),   # Input image
   ...
   layers.Dense(1) ]), # Output probability of image being true
```

```
def generator_loss(fake_output):
   # compares the discriminator's predictions on fake images to 1
   return BCE(tf.ones_like(fake_output), fake_output)
```

```
def discriminator_loss(real_output, fake_output):
   # compares the discriminator's predictions on real images to 1
   real_loss = BCE(tf.ones_like(real_output), real_output)
   # compares the discriminator's predictions on fake images to 0
   fake_loss = BCE(tf.zeros_like(fake_output), fake_output)
   total_loss = real_loss + fake_loss
   return total_loss
```

```
for epoch in range(Nepoch):

   generated_images = generator(noise) # generate fake image
   fake_output = discriminator(generated_images) # probability of image being true (should be low!)

   real_output = discriminator(real_images) # probability of image being true (should be high!)

   disc_loss = discriminator_loss(real_output, fake_output) # loss
   gen_loss  = generator_loss(fake_output)

   gen_gradients = gradient(gen_loss, generator.trainable_variables) # gradient
   disc_gradients = gradient(disc_loss, discriminator.trainable_variables)

   generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables)) # update weights & biases
   discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
```

open neural-network-part5.ipynb

Generative models – Diffusion

$$x_t \sim \mathbb{N}\left(\sqrt{\frac{T-t}{T}(1-\alpha)x_{t-1}}, \alpha + \frac{t}{T}(1-\alpha)\right)$$

Diffusion process: gradually transforming random noise into the data and train the network to learn how to reverse the process to recover the data

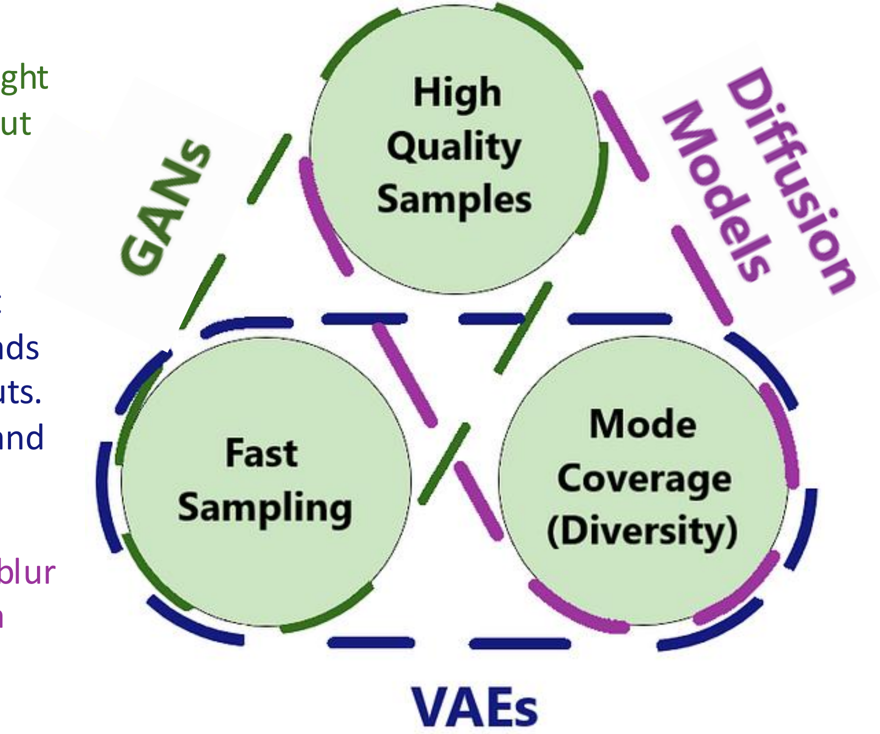
Reverse process: $x_{t-1} \sim \mathbb{N}(\mu(x_t), \sigma^2(x_t))$, where μ and σ^2 can be learnt using neural network

Comparison

GAN: When the discriminator has overtrained, the generator might be happy enough to produce a small part of the data diversity but with high fidelity. It is also often difficult to train or decide convergence due to the adversarial nature.

VAE: Because the encoder predicts the distribution of the latent space, the overlap between the distribution of similar inputs leads the optimal decode to their average, which often blur the outputs. But it has high diversity due to the relatively large latent space and often easy to train.

Diffusion: Diffusion process often requires ~1000 steps to fully blur the input image and reach white noise, and therefore requires a more complicated network which makes sampling slow.



prerequisite

We will start at 1:05pm, before that please

git clone/pull from https://github.com/qyx268/astr4004/

Install emcee, dynasty, corner
Install torch, sbi