Introduction to Quantum Computing

Dr. Rahul Remanan

Monte Carlo simulations

09/19/25

Monte Carlo simulations

Monte Carlo simulations are a series of approximation techniques using random sampling, to understand and quantify systems numerically; especially complex ones with very poorly understood numerical processes or tools to model them.

Monte Carlo simulations

- Ability to design complex system behaviors with minimal observation points
- Correlate really well with the target system behavior
- Simple and easily scalable computational steps

Monte Carlo simulations

Only prerequisites:

- Sampling should be truly random
- Each sample drawn independently
- Ensures that the probability distribution of the randomly sampled population matches the target system behavior that it aims to model

Modeling single coin tosses

Matching the sampling and target probability distributions, for a coin toss:

- Two faces with equal probability:
  - $(p_{head}, p_{tail} = 0.5)$
- Each toss independent of the previous one
- `print('Head' if random.SystemRandom().random() > 0.5 else 'Tail')`

Modeling two simultaneous coin tosses

- Probability of coin 1 returning a head or a tail:
  - $(p_{coin\ 1\ head}, p_{coin\ 1\ tail} = 0.5)$
- Probability of coin 2 returning a head or a tail:
  - $(p_{coin\ 2\ head}, p_{coin\ 2\ tail} = 0.5)$
- Each coin tosses are independent of the other

Modeling two simultaneous coin tosses

- Probability of a head given the other outcome is a tail or probability of a tail given the other outcome is a head:
  - $(p(head|tail) + p(tail|head)) = (p_{coin\ 1\ head} * p_{coin\ 2\ tail}) + (p_{coin\ 1\ tail} * p_{coin\ 2\ head}) \rightarrow p(head|tail) + p(tail|head) = 0.5$
- Probability of a head given the other outcome is a head:
  - $(p(head|head)) = (p_{coin\ 1\ head} * p_{coin\ 2\ head}) \rightarrow p(head|head) = 0.25$
- Probability of a tail given the other outcome is a tail:
  - $(p(tail|tail)) = (p_{coin\ 1\ tail} * p_{coin\ 2\ tail}) \rightarrow p(tail|tail) = 0.25$

Modeling two simultaneous coin tosses

- Probability of at least one head:
  - $(p(head|tail)\ or\ p(tail|head)\ or\ p(head|head)) = (p_{coin\ 1\ head} * p_{coin\ 2\ tail}) + (p_{coin\ 1\ tail} * p_{coin\ 2\ head}) + (p_{coin\ 1\ head} * p_{coin\ 2\ head}) \rightarrow p(head|tail) + p(tail|head) + p(head|head) = 0.75$
- Probability of at least one tail:
  - $(p(head|tail)\ or\ p(tail|head)\ or\ p(tail|tail)) = (p_{coin\ 1\ head} * p_{coin\ 2\ tail}) + (p_{coin\ 1\ tail} * p_{coin\ 2\ head}) + (p_{coin\ 1\ tail} * p_{coin\ 2\ tail}) \rightarrow p(head|tail) + p(tail|head) + p(tail|tail) = 0.75$

Modeling regular unweighted dice rolls

Requirements :

- Six dice faces
- Equal probability of the dice faces
- Each dice roll independent of the previous ones

Modeling regular unweighted dice rolls

- $p_{face} = 1/6$

Expected value

$$E[X] = p_{face} \sum_{i=1}^{6} i$$

$$E[X] = \frac{(1+2+3+4+5+6)}{6}$$

$$\therefore E[X] = \frac{7}{2}$$

Modeling regular unweighted dice rolls

Variance

$$Var(X) = p_{face} \cdot \sum_{i=1}^{6} (i - E[X])^2$$

$$Var(X) = \frac{1}{6} \cdot (\sum_{i=1}^{6} (i - \frac{7}{2})^{2})$$

$$Var(X) = \frac{1}{6} \cdot ((\frac{-5}{2})^2 + (\frac{-3}{2})^2 + (\frac{-1}{2})^2 + (\frac{1}{2})^2 + (\frac{3}{2})^2 + (\frac{5}{2})^2)$$

$$\therefore Var(X) = \frac{35}{12} = 2.91\bar{6} \approx 2.92$$

Modeling regular unweighted dice rolls

Standard deviation (σ)

$$Var(X) = \sigma^2$$

$$\therefore \sigma = \sqrt{Var(X)} = \frac{\sqrt{35}}{2\sqrt{3}}$$

Modeling regular unweighted dice rolls

Probability mass function

For an n faced dice, it can be viewed as a discrete random number generating function $f(X)$; with the corresponding probability mass function $P(X)$ denoted as follows:

$$x_1 \longmapsto p_1, x_2 \longmapsto p_2, x_3 \longmapsto p_3, \ldots, x_n \longmapsto p_n \implies 1 \longmapsto p_1, 2 \longmapsto p_2, 3 \longmapsto p_3, \ldots, n \longmapsto p_n$$

Where: $p_i = \frac{\text{Total number of appearances of the } i^{th} \text{ dice face}}{\text{Total number of dice throws}}$

Modeling regular unweighted dice rolls

Expected value

$$\mu = \sum_{i=1}^{n} p_i \cdot x_i$$

$$\therefore \mu = \sum_{i=1}^{n} p_i \cdot i$$

Modeling regular unweighted dice rolls

Variance

$$Var(X) = \sum_{i=1}^{n} p_i \cdot (x_i - \mu)^2$$

Modeling regular unweighted dice rolls

Standard deviation

$$\sigma = \sqrt{Var(X)}$$

Modeling a game of fair roulette

- Monte Carlo simulation performed using the `FairRoulette()` class
- Mimics the behavior of a roulette wheel with 36 slots, each uniquely numbered between 1 through 36
- When a bet corresponding to a pocket numbered between 1 and 36 is made, the `spin` function in the `FairRoulette()` class returns a positive value corresponding to the (***betting amount * pocket odds***) , where: $pocket\ odds = 35$; to represent the amount of money won or returns a negative value corresponding to the ***betting amount***, to represent the amount of money lost; while playing the game of roulette

History of Monte Carlo simulations

Buffon's needle problem:

- Formulated by Georges-Louis Leclerc, Comte de Buffon in the 18th century
- What is the probability of a needle of length (l) that is falling between parallel lines of even spacing (d), to intersect one of the lines; where: $l < d$?
- One of the oldest mathematical problem involving random sampling and geometric probability.

History of Monte Carlo simulations

Buffon's needle problem:

- Non-obvious convergence to an irrational constant, from just a series of random observations
- Value of Pi emerges from this seemingly random sampling process:
  - $p_{needle\_intersection} = 2.length_{needle} / \pi.distance_{lines}$
- Solution is closely linked to the angle of rotation of the needle

History of Monte Carlo simulations

Estimation of pi using random points inside a square:

- Probability of a random point inside a semi-circle of radius r, contained inside a square of side r; with the center of the semi circle lying on one of the edges of the square:
  - $p_{pi\ estimation} = \pi / 4$

Error estimation using Riemann-Remanan sentinel sampling

- Sentinel area defined as an area of the largest square contained inside the semi circle that is $^{1}/_{4}{}^{\text{th}}$ the segment of a circle of radius r

Error estimation using Riemann-Remanan sentinel sampling

- Probability of the random point lying with in the semi circle to be also contained with in the sentinel area:
  - $p_{semi\ circle\ sentinel} = {}^{2}/_{\pi}$
- Probability of the random point lying inside the sentinel area:
  - $p_{sentinel} = {}^{1}/_{2}$
- Sampling error:
  - $$E_{Riemann\text{-}Remanan} \in [\min\{p_{sentinel} * 2, p_{semi\ circle\ sentinel} * p_{pi\ estimation} * 2\}, \max\{p_{sentinel} * 2, p_{semi\ circle\ sentinel} * p_{pi\ estimation} * 2\}]$$

Riemann-Remanan error correction

- Riemann-Remanan probabilistic error correction works by performing stochastic integration over a sentinel population, which returns an error estimate of the random sampling process
- An output range is computed using the error estimation to compensate for any potential sampling errors
- The final output is generated using a final stochastic sampling step using the output range

History of Monte Carlo simulations

Neutron nucleus interactions:

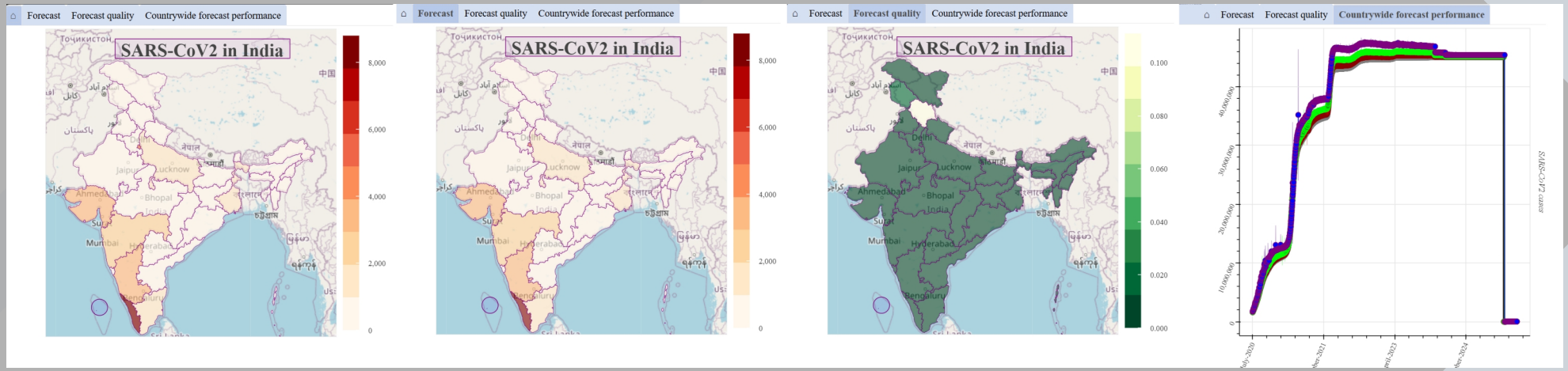
- Enrico Fermi first proposed the use of random sampling to model neutron-nucleus interactions
- Further expanded by US-Polish physicist Stanislav Ullam and US-Hungarian physicist and computer scientist John Von-Neumann
- The name Monte Carlo simulation was created to obfuscate the random sampling technique used by the United States to model the neutron-nuclei interactions for the development of the atomic bomb

Monte Carlo simulations

Modeling public health:

- Monte Carlo simulations using deep-neural networks called deep Bayesian networks
- Uses a variant of the deep-neural networks called deep neural networks with Bayesian approximation that uses Markov Chain Monte Carlo simulations
- Probabilistic modeling of the infectious disease dynamics of large populations

Monte Carlo simulations in healthcare



Monte Carlo simulations in digital computing

- Large number of numerical computing techniques and software programs
- Modeling probability distributions are very straightforward
- Modeling random independent events is challenging

Limitations of randomness in digital computing

- Python supports API calls for randomness (`random` and `random.SystemRandom()`)
- Python uses the Mersenne Twister as the core generator (`random.random()`)
- Python also uses operating system (OS) collected entropy pool in Linux (`random.SystemRandom().random()`)

Mersenne Twister pseudo-randomness

- Produces 53-bit precision floats and has a period of 2**19937-1
- Underlying implementation in C is both fast and threadsafe
- One of the most extensively tested pseudo-random number generators

Python random.SystemRandom()

- `SystemRandom` class uses the system function `os.urandom()`
- Generate random numbers from the OS sources
- Outputs a uniform probability distribution using the function call: `random.SystemRandom().random()`

Mersenne Twister pseudo-randomness

- Completely deterministic
- Unsuitable for applications that require true randomness
- Not for cryptography or accurate Monte Carlo simulations

Python random.SystemRandom()

- Closer to true randomness and works for most random sampling use cases, such as: for cryptography in digital computing
- Dependent on the underlying entropy collection process by the OS
- Lacks speed and robustness

Python `random.SystemRandom()`

- OS derived entropy collection, making the code-base behavior highly OS platform dependent; despite successful code execution
- For Linux:
  - Availability dependent on `getrandom()` syscall
- For Windows:
  - Using Windows `BCryptGenRandom()`

Linux getrandom() syscall

- Used in blocking mode, until initializing system `urandom` entropy pool of 128-bits collected by the kernel
- Non-blocking mode can be enabled using the `GRND_NONBLOCK` flag
- PEP 524, with potential to use hardware derived randomness sources such as CPU thermal sensor linked random number generation

Microsoft Windows BCryptGenRandom()

- Random number generation using deterministic random bit generators
- NIST SP800-90 standard
- Highly deterministic and deviates significantly from the Linux implementation

Monte Carlo simulations in quantum computing

- Faster than pseudo-random sampling in traditional digital computers
- Hardware linked entropy collection that is highly parallelizable
- Independent of any underlying OS processes

Monte Carlo simulations in quantum computing

- True randomness through stochastic quantum processes
- Minimizing errors and biases due to faulty sampling commonly associated with the sampling techniques in traditional digital computer
- Meets the criteria of independent random sampling and makes it truly representative of the population it is drawn from

Quantum processes for randomness

- Quantum random number generators
- Photon walk
- Thermodynamically bound computational states

Predictive non determinism

- Probabilistic systems, with no definite outcome over the course of the system being observed
- Each outcome has a prior likelihood of appearance
- The observation of an outcome of the system is the only available process of determining the exact system behavior at that particular instance and the description of a hermetically sealed probabilistic system by an outsider is only through a probabilistic description of the outcome (causal uncertainty approach)

Predictive non determinism

Coin toss outcomes of a single non weighted fair coin:

- Most likely outcome: either a head or a tail:
  - $p=0.5$
- Having a head or a tail have equal likelihood
- By modeling the system as probabilistic instead of having a deterministic outcome, helps describe the system behavior more accurately

Predictive non determinism

Coin toss outcomes of two non weighted fair coins:

- Most likely outcome is the appearance of one head and one tail:
  - $p=0.5$
- Appearance of both heads or both tails have equal, but lower likelihood:
  - $p=0.25$
- Best guess for the two coin toss outcome without observing the outcomes, is the most likely outcome; where there is one head and one tail, but this does not imply that there is only one possible outcome

Causal uncertainty

- Modeling the world as fundamentally not completely predictable one is a simple, yet valid solution to predictive non determinism
- Included in the Copenhagen doctrine, proposed by Neils Bohr and Werner Heisenberg
- Based on Max Born's statistical interpretation of the wave function (ψ)

Causal uncertainty

- System with multiple possible outcomes ($Ks_{outcome}$), represented by the function: $f(x)$
- Knowledge of $f(x)$ outcome:
  - Final state of the system Ks_{final}
  - Without observation represented by the function: $k(f(x))$
  - With observation represented by: $k_o(f(x))$
- Corresponding probability mass functions:
  - $k(f(x)) \mapsto P^k(k(f(x)))$
  - $k_o(f(x)) \mapsto P^{k_o}(k_o(f(x)))$

Causal uncertainty

$$Ks_{final} \subset Ks_{outcome}$$

$$k_o(f(x)) \subset k(f(x)) \subset Ks_{outcome}$$

$$Ks_{final} \stackrel{?}{=} k_o(f(x)) \Rightarrow \text{observer effect}$$

$$Ks_{outcome} \stackrel{?}{=} k(f(x)) \Rightarrow \text{hidden states / information asymmetry}$$

$$|k(f(x))| > 1 \text{ and } 1 \notin P^k(k(f(x))) \Rightarrow \text{super position}$$

$$|Ks_{final}| \stackrel{?}{=} 1 \text{ and } |k_o(f(x))| \stackrel{?}{=} 1 \Rightarrow \text{stable / unstable outcomes}$$

$$1 \in P^{k_o}(k_o(f(x))) \text{ or } 1 \notin P^{k_o}(k_o(f(x))) \Rightarrow \text{collapsing super position / observation uncertainty}$$

Causal uncertainty

- Both $k(f(x))$ and $k_o(f(x))$ are equally valid, mutually exclusive descriptors of the $f(x)$ outcome
- Causal uncertainty is the description of $f(x)$ outcomes only using $k(f(x))$
- Does not violate the physical laws of the universe or require super position of the actual outcomes, since $k(f(x))$ is just the superposition of the knowledge of all possible outcomes and $k(f(x))$ can only exist without observing the outcome associated with the system $f(x)$ and can be called the Landauer's principle update to the Copenhagen doctrine

Schrodinger’s cat

- Whether or not a radioactive atom in the sealed box has undergone radioactive decay, which triggers a poison release; killing the cat inside the sealed box
- Quantum phenomenon bound large scale system state behavior
- Internal state of the system (alive or dead cat) is determined quantum probabilistic behavior

Schrodinger’s cat

- With in the box, the system has acquired an internal state out of two possibilities
- An outside observer does not have any means to determine what that internal system state is, without unsealing the box
- The system can only be described through probabilistic methods without actually observing the internal state

Infinite number of Schrodinger's cats

- Variation of Schrodinger's cat thought experiment
- Infinite number *(large N)* of Schrodinger's cats
- Ratio of total number of alive / dead cats is always a constant

Infinite number of Schrodinger's cats

- Probability of radio active decay:
  - $p_r = -\max\{-(t / (2 * t_{1/2})), -1\}$
- Ratio of total number of alive / dead cats:
  - $(N - N * p_r) / N * p_r = (1 - p_r) / p_r$
- Easily predictable to an outside observer, just with the knowledge of the radioactive decay rate, where half life $(t_{1/2})$ is the amount of time elapsed to have ½ likelihood that an atom underwent radioactive decay

Remanan's ugly sweater wearing cat

- There is a hermetically sealed box that an observer cannot look inside

- Setup inside the box consisting of the following:
  - A cat not wearing an ugly sweater
  - A sweater robot that can make the cat wear an ugly sweater
  - A fair coin toss machine that controls the robot

- The cat is made to wear an ugly sweater every time the robot gets a head on the coin toss machine

Remanan’s ugly sweater wearing cat

- Can we describe what is happening inside the system without observing the internal state of this probabilistic system?
- What happens when we observe the internal state of such a system?
- What changes when an observation is made about the internal state of such a system?

Remanan’s ugly sweater wearing cat

- Local stochastic phenomenon bound global system state behavior
- Internal state of the system (cat wearing an ugly sweater or not) determined by whether or not a fair coin toss returns a head
- For an outside observer, there is an uncertainty about the internal state without actually observing the internal state

Infinite number of Remanan's ugly sweater wearing cats

- Variation of Remanan's ugly sweater cat thought experiment
- Infinite number *(very large N)* of Remanan's ugly sweater wearing cats
- Ratio of total number of sweater wearing cats / non sweater wearing cats (always a constant *(1)*) becomes easily predictable to an outside observer, just with the knowledge of the likelihood of the probabilistic system's outcome (likelihood of the appearance of a head in a fair coin toss)

Predictive non determinism

- Probabilistic systems do not violate physical reality of the universe
- System acquires an internal state, but predicting that state will be always probabilistic without observing the actual state
- System behavior can be explained without the need for exotic ideas such as super position of the internal state of the system (cat is not in a super position inside the box in Schrodinger's cat thought experiment or in the Remanan's ugly sweater cat thought experiment)

Causal uncertainty

Solution to the *“God doesn’t play dice problem”*:

- Internal state of the system and the knowledge of the internal state of the system are two distinct physical phenomenon
- Knowledge of the internal state requires the acquisition of information about the system
- Information itself is a physical phenomenon with real and distinct energy costs associated with its storage and retrieval (Landauer’s principle)

Causal uncertainty

Solution to the *“God doesn’t play dice problem”*:

- In both the Schrodinger’s cat thought experiment and the Remanan’s ugly sweater wearing cat thought experiment, knowledge of the internal state of the system exists in super position; without observation
- The physical reality of the internal state is not in a super position
- Both the Schrodinger’s cat thought experiment and the Remanan’s ugly sweater wearing cat thought experiment are just problem statements for describing the behavior of such similar probabilistic systems

Information asymmetry

- Another advantage of using causal uncertainty model is to deal with information asymmetry
- The state where there is only partial information available about the system that is modeled
- Treating the system as not completely predictable helps make better decisions about the system behavior through predictive non determinism, by describing the system using the most likely outcomes as an ordered multiset

Bayes theorem

$$p(H|E) = p(H) * p(E|H) / p(E)$$

- Referred to as the equation of common sense, works by updating our beliefs about a probabilistic system; once new information is available
- Invented by Thomas Bayes and the current mathematical notation was independently formulated by Pierre-Simone Laplace
- Computes posterior probability $(p(H|E))$, of a hypothesis (H) given the evidence (E)

Bayes theorem

Components of Bayes theorem:

- $p(H)$ → Prior probability (cumulative probability of the hypothesis)
- $p(E|H)$ → Probability of the evidence given the hypothesis is true
- $p(E)$ → Probability of the evidence

Bayes theorem

The subway train problem:

- Cumulative probability of getting hit by a subway train:
  - ***(p(H), probability$_{hypothesis}$)*: *One in a million (10^{-6})***
- Probability of a passenger voluntarily crossing the yellow line:
  - ***(p(E), probability$_{evidence}$): One in a hundred (10^{-2})***
- Probability of the passenger having voluntarily crossed the yellow line when being hit by a subway car:
  - ***(p(E|H), probability$_{evidence\ given\ the\ hypothesis}$): Ninety-nine out of hundred (0.99)***

Bayes theorem

- Probability of getting hit by a subway train, given a passenger voluntarily crossing the yellow platform line:
  - ***(p(H|E), probability$_{hypothesis}$ given the evidence): 99 times greater risk than the cumulative probability of getting hit by a subway train***

Deep Bayesian networks

- Incorporates stochastic behavior into deep neural networks
- Parameterized non-linear function approximators similar to deep neural networks, but those that generate a distribution of predictions; by using a prior distribution of weights
- Quantifies prediction uncertainty by measuring the standard deviation of the predicted outputs

Deep Bayesian networks

- Not a true universal function approximator due to limitations of the surjective nature of machine learning (Cantor's paradox)
- Machine learning models, even the Bayesian ones do not have to satisfy the criteria of a true universal function approximator, to be useful
- Since the most commonly encountered phenomena in the universe can be treated as just a smaller subset of the universe of all the mathematical operations, machine learning models can be applied to approximate these phenomena very successfully

Deep Bayesian networks

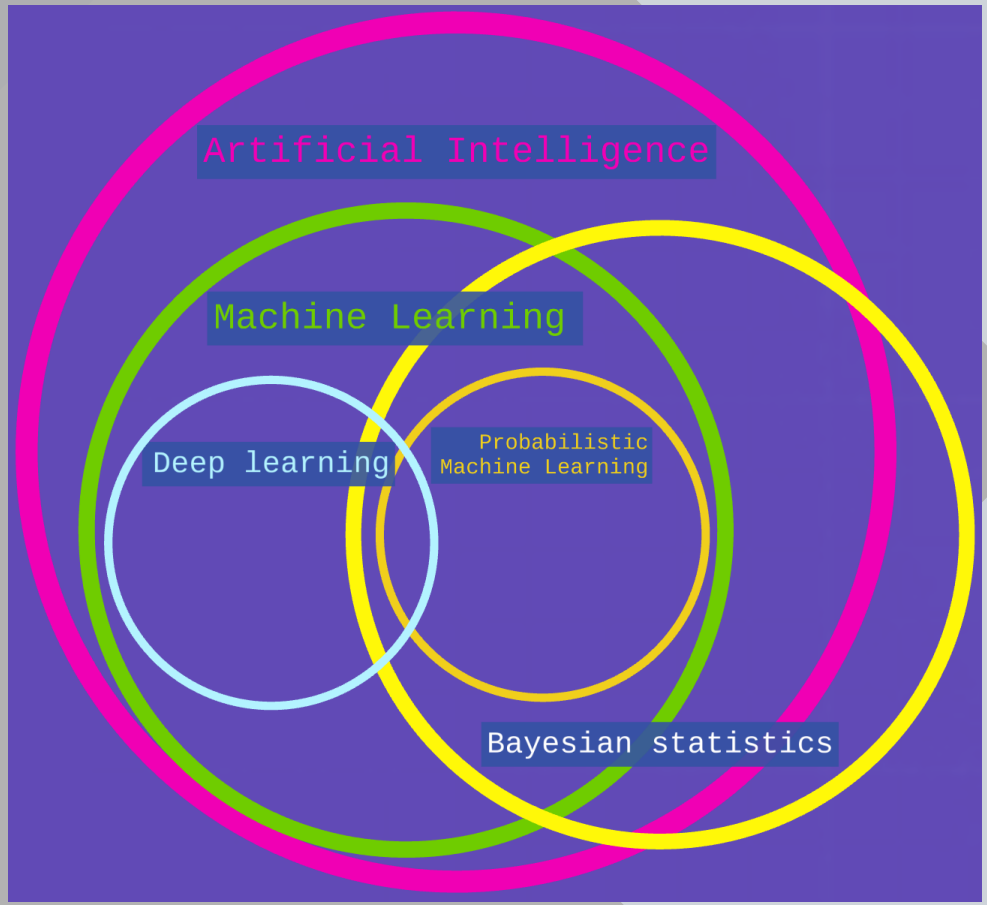
- Deep neural network based approach to dealing with a world that is filled with uncertainties
- An automated mechanism to quantify the uncertainties associated with a prediction, helps make a better decision around the predicted outcome
- Modeling the world as a fundamentally not completely predictable one, also incorporates the idea of causal uncertainty

Stochasticity in neural networks

- Stochastic auto-differentiation
- Input perturbations (Data augmentation) and model perturbations (Deep Bayesian networks)
- Training (stochastic weights and layers), inference (Bayesian and model ensembling) and optimization (hyper-parameter tuning)

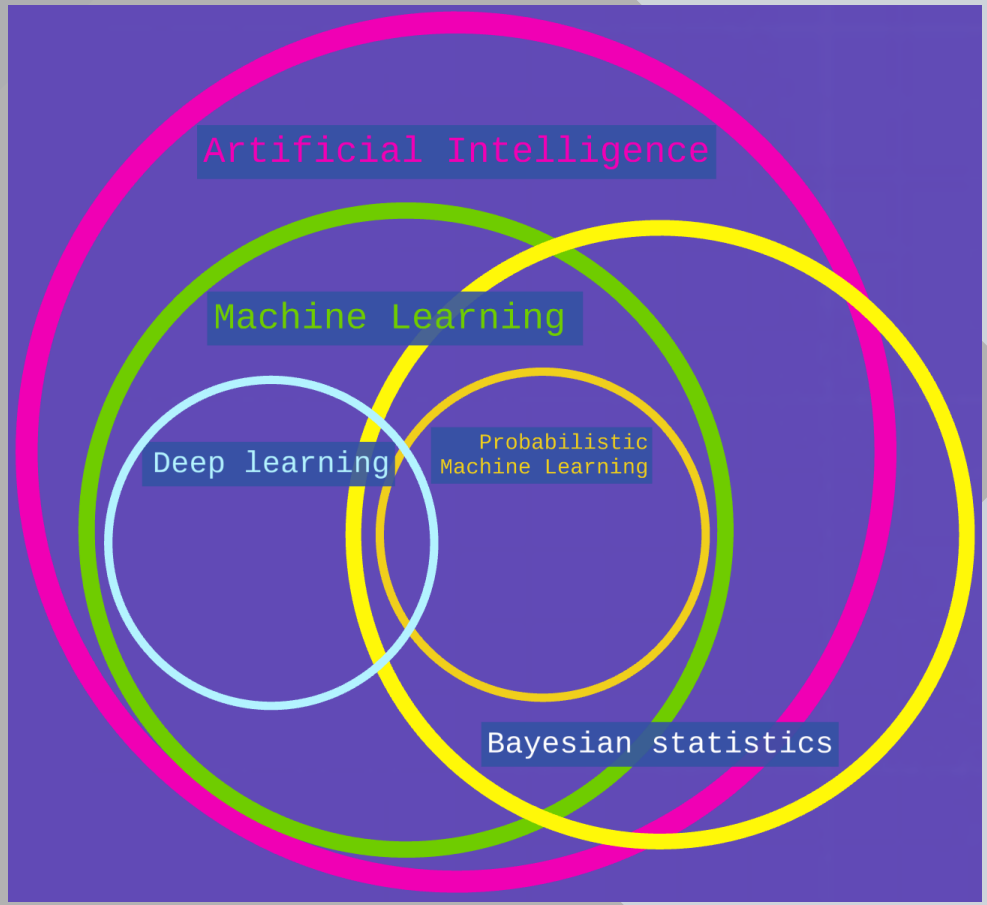
Stochasticity in neural networks

- Mechanism to quantify model uncertainty, resulting in better automated decisions; incorporating causal uncertainty
- Model uncertainty as a proxy measure for quantifying risk
- Pooling uncertainty measures through model ensembling to improve prediction robustness



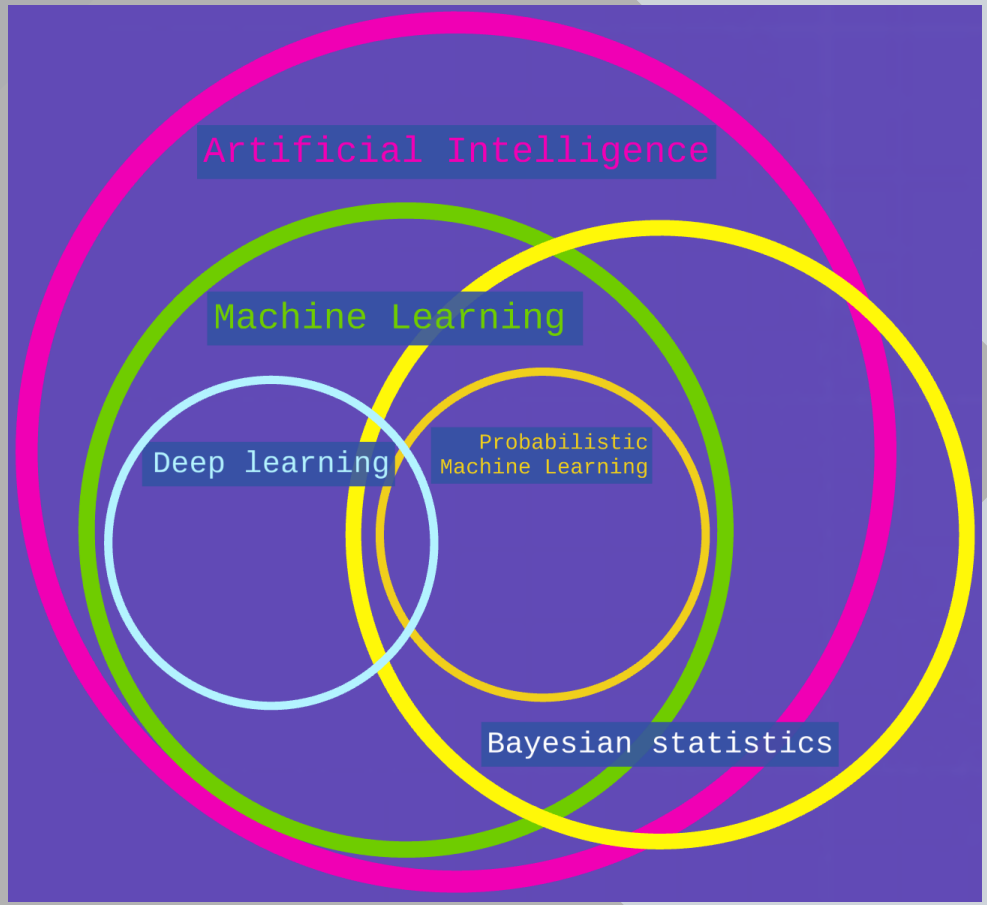
Stochasticity in neural networks

- Development of explainable artificial intelligence through the quantification of model uncertainty
- Safer models through improved resilience against adversarial actors and poisoned models
- Addressing deep neural network's over confident false predictions



Stochasticity in neural networks

- Counter-intuitive notion of giving more freedom to the learning algorithm to make it more controllable
- Quantifying model uncertainty helps develop tools and methods to reason about the sources of uncertainty
- Creating more effective learning processes such as curriculum learning, with less risk of being stuck in a local minima



Quantum processes linked deep neural networks

- Quantum processes linked random sampling as a proxy for interfacing deep neural networks with the quantum hyperspace
- Quantum stochasticity in data augmentation strategies
- Quantum stochasticity guided model training, model optimization, Bayesian inference, model ensembling and error correction