The original article :
https://towardsdatascience.com/operator-learning-via-physics-informed-deeponet-lets-implement-it-from-scratch-6659f3179887
The original deepONet paper :
https://arxiv.org/abs/1910.03193

ODE and PDE :

are tools used to describe physical systems and processes,
capturing the **continuous change** of quantities over time and space.
They don't take just single values as inputs, they **take functions.**
They are tackled using :

- **Numerical solvers :** For every input function, the solver must be run all over again.
- **DeepONet (Deep Operator Netwok) :** Aims to predict the output function without having to re-run a numerical solver each time.

Physics-Informed Learning :

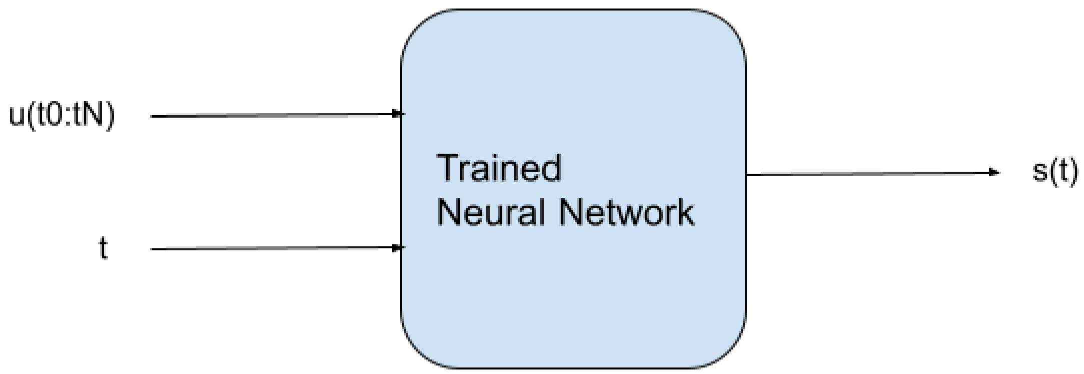
is a branch of machine learning,
combines the physical principles with data science.
We are no longer just asking our model to **learn from data**, we are **guiding it with principles** derived from centuries of scientific inquiry.

Physics-Informed DeepONet :

★ **DeepONet :** designed to map entire functions to other functions

u(.) - - - - -	operator G	- - - - - > s(.)
u(.) - - - - -	trained Neural Network	- - - - - > s(.)

For our case study :



The value of s(t) depends on :

- the value of s(.), which depends on the value of u(.)
- at which time instance the s(.) is evaluated

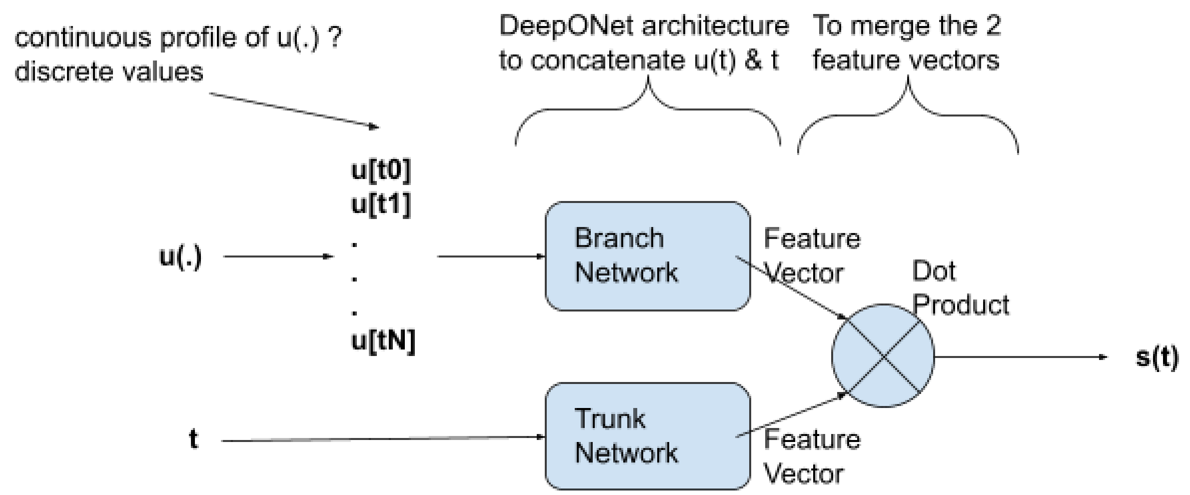
How should we input a **continuous profile of u(.)** to the network ?

- we represent u(.) **discretely** : evaluate u(.) values at sufficient but finite many locations : u[t0], u[t1], …, u[tN]
  ⇒ Those locations are referred to as sensors in the DeepONet original paper.

How should we **concatenate** the input t and u(.) ?

- ***DeepONet*** proposed a new network architecture for performing operator learning :
    - **a branch network :** takes the discrete function values as inputs & transform them into a **feature vector.**
    - **a trunk network :** takes the coordinate t & converts it into **feature vector.**

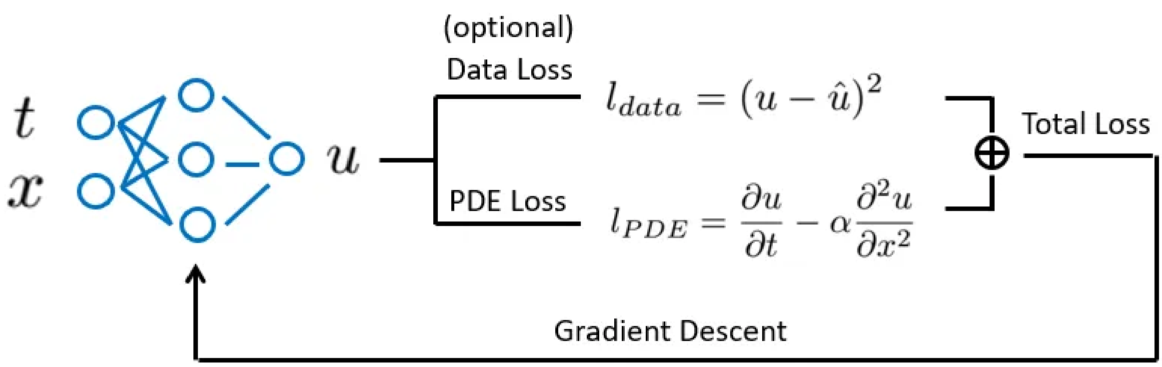
- The two feature vectors are then **merged** by **a Dot Product.**
- **The end result** is used as the **prediction of s(.) evaluated at the input coordinate.**



Limits of DeepONet :
May not generalize well, especially when faced with input functions that lie outside the distribution of its training data. I needs a large amount od data for training to remedy that ! ⇒ expensive & time consuming.

- **PINNs :**

a type of Neural Network
the network is trained to **fit the data** & respect the known **physical laws described by Diff. Eq.**
Achieved by introducing a **"ODE/PDE loss"** :
measures the degree of violation of the governing diff. eq.
⇔ measures **if the predicted solution satisfies the governing diff. eq.**
⇒ we inject the **physical laws** into the **network training process** to make it physically informed :



Limits of PINNs :
are typically trained for specific input parameters.
⇒ whenever the input parameters have changed, we would need to retrain the PINNs !
⇒ Not particularly effective for real-time inference under different operating conditions.
=> but the DeepONet can handle varying input parameters

★ **PIDeepONets :**

combines the strengths of :
DeepONets (efficiency : real time inference) & PINNs (accuracy : consistency)

- takes a **function as an input** & produces a **function as an output**
  ⇒ makes it efficient for real-time inference new input functions **without retraining**
- incorporates known physical laws into its learning process.
  ⇒ these laws are introduced in the **loss function** during training.