PINNs - Physics Informed Neural Networks

Tensorflow, PyTorch - two things to study about

Neural Networks are almost every where right now, even ChatGPT is a big neural network with billions of parameters

Why neural networks are so good is because of how they approximate functions easily, and this is also one of the reasons they also overfit the training dataset.

Thus we need lots of data along with it some clean batching to be done

We cannot use Neural Networks when the amount of data we have is not large/big enough, but then what is big enough number, well it depends on the model and parameters

For example if we have a 2 hidden layered NN, with 64 neurons in each layer, 10 inputs and 1 output,

then N samples = k* N parameters

k = 10∗64 + 64∗64 + 64∗1 (break up into layers multiply each layer with its previous one) = approx 5000, and total samples then = 50,000

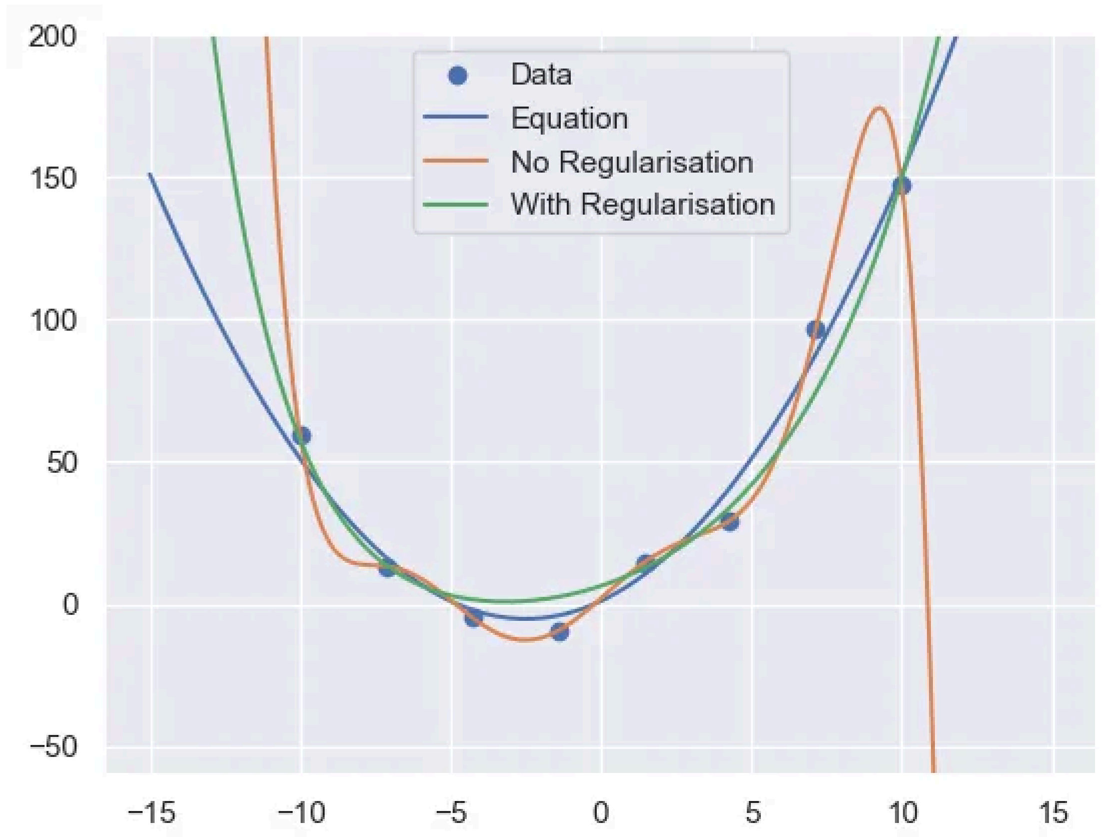
So this k is also known as regularization

Loss fn for NN is as follows

A common approach to avoid overfitting is regularisation. You can do regularisation with neural networks just like you would do it with linear regression,

$$Loss_{reg} = \frac{1}{N}\sum_{i}^{N}(f(x_i|\theta) - y_i)^2 + \lambda||\theta||_2^2$$

Thus we minimize the weights, so that no weight is absurdly high, giving a less chunky function.



To add a Physics-informed priors in NN, what we do is we do it by imbeding the network with information in the form of a DE, So again we take the mean squared error of the physics loss, as MSE or even RMSE, is useful because of its convex shape and also penalizing values with having larger losses, having a minima at some point or the other, and thus now adding this physics loss as a DE function, we get this

> Say we have a differential equation *g(x, y)* = 0, some data *{x_j, y_j}* and a neural network *f(x* | *θ)* that approximates *y*. For a PINN, we would get a loss function that looks like the following,

$$Loss_{PINN} = \underbrace{\frac{1}{N}\sum_{j}^{N}||f(x_j|\theta) - y_j||_2^2}_{\text{Data loss}} + \lambda\underbrace{\frac{1}{M}\sum_{i}^{M}||g(x_i, f(x_i, |\theta))||_2^2}_{\text{Physics loss}}$$

here the differential eqn is g where each g(x,y)=0 and thus each point if neural network diverges from predicitng the y, the value of differential equation will be some non-zero and will be our physics error, the neural network approximating y which we predict is the function f, for given parameters xi and yi,

and there is also this lambda term, which regulates how much physics loss needs to be considered and the machine model based on epochs, can minimze the loss, relative strength of data loss function and physics loss function is lambda

Now how do we conver this differential equation to integrate in our PINNs, let’s take a simple example of cooling of a hot coffee cup, it obeys simple law of physics

$$\frac{dT(t)}{dt} = r(T_{env} - T(t))$$
$$T(t) : \text{temperature}$$
$$T_{env} : \text{temperature of the environment}$$
$$r : \text{cooling rate}$$

Making a PINN of this eqn would be something like this

$$g(t,T) = \frac{dT(t)}{dt} - r(T_{env} - T(t)) = 0$$
$$g(t, f(t|\theta)) = \frac{df(t|\theta)}{dt} - r(T_{env} - f(t|\theta))$$
$$Loss_{PINN} = \underbrace{\frac{1}{10}\sum_{j}^{10}(f(t_j|\theta) - T_j)^2}_{\text{data loss}} + \lambda\underbrace{\frac{1}{M}\sum_{i}^{M}\left(\frac{df(t_i|\theta)}{dt_i} - r(T_{env} - f(t_i|\theta))\right)^2}_{\text{physics loss}}$$

for using the derivate of the neural network, we have torch.autograd module which has a function called as grad() with having create_graph = True

PyTorch and Tensorflow is what we need to explore now to inculcate the PINNs