Physics-informed neural networks for Burges equation, 2D Navier-Stokes equation and several playground demo

zjr

August 13, 2023

Abstract

use pytorch to solve Physics-informed neural networks for Burges equation, 2D Navier-Stokes equation and several playground demo

1 the idea of pinn and the simplest demo

1.1 rethink the idea of the pinn

pinn, physics-informed neural networks is an idea to use nerual networks to solve mathematical and physical equations.

1.1.1 essence of the neural network

As people who be familiar with all kinds of neural networks all know, the true essence among all the variety that a neural network can do is gradient descent, which could be a unified method to approximate all kinds of complicated functions

To do the gradient descent we calculate, in some sense, the loss between the function we want to approximate and the function the neural network it self as.

Formally speaking, it take the true function denoted as f_{true} ans $f_{nerualnet}$ and create a proper loss function $loss$ between them, which quantified their difference in to a non-negative real number.

$$loss : (f_{true}, f_{nerualnet}) \to r \in \mathbf{R}$$

We then calculate the gradient of the $loss$ respect to the $f_{nerualnet}$

$$\nabla loss = \frac{\partial loss}{\partial f_{nerualnet}}$$

And we can recursively compute the better $f_{nerualnet}$

$$f^{n+1}_{nerualnet} = f^{n}_{nerualnet} - lr\nabla loss$$

The lr is represent the learning rate of the neural network

1.2 Physics equations as new supervisor

The conventional neural networks are directly supervised by the given function f_{true}(often represent by large amount of input-output data pair).

For some other function that we are not know the output but we know some property that the function has.

As in physics context the function we want solve are hard to directly derive the solution, but we do know the differential equation the function satisfies.

1.3 first demo the mass spring with damping

1.3.1 the physics model and the analytic solution

the function we want to solve is the x verse time function, $x(t)$. the equilibrium is set to be $x = 0$ the initial state includes the velocity $v(0) = \dot{x}(0)$ and the initial amplitude $x(0)$

use the newton second law we can build the differential equation for the model

$$F = ma = m\ddot{x}$$

$$m\ddot{x} = -kx - \mu\dot{x}$$

where k is the coefficient of hooks law, μ is the air friction
and we can rewrite it in homogeneous form as follow

$$m\ddot{x} + kx + \mu\dot{x} = 0$$

it can be solved easily by mathematical method
and the solution is as follow

$$x = Ae^{-\frac{\mu}{2m}(t+\phi)}cos(\omega(t+\phi))$$

$$\omega = \sqrt{\frac{k}{m} - \frac{\mu^2}{4m^2}}$$

$$A = \frac{x(0)}{e^{-\frac{\mu}{2m}\phi}cos(\omega\phi)}$$

1.3.2 model set up

we use FCN(fully connected network),with one dimension input, one dimension output, hidden-layer size of 16 and 6 hidden-layer

the loss function contains two parts
the initial-condition part $loss_{init}$ and the physics-informed part $loss_{phy}$
set out FCN to be a function
we want $F(t)$ to be fit with the function $x(t)$
so the function $F(t)$ should also fit the differential equation mentioned above

$$loss_{init} := (F(0) - x(0))^2$$

$$loss_{phy} := (m\ddot{F} + kF + \mu\dot{F})^2$$

$$loss_{tot} := loss_{init} + loss_{phy}$$

1.3.3 code and result

the code of this project are on github https://github.com/unjerry/pinn_playground/tree/pinndemo, and the result are in the gif picture in the same folder as the article "testforautograd0-5.gif" and "testforautograd.gif"