mcTangent - Wesley Lao

wesleygarlao

November 2022

1 To Do

1. Repository: https://github.com/wglao/mcTangent
2. Reproduce problem 1
3. Write down all possible ideas (your notes):
    - CNN architecture
    - Runge-Kutta methods
        - adaptive time steps as in RK45, taking advantage of mcTangent's ability to be applicable to many time discretization schemes - can learn time step
    - implicit or semi-implicit method
    - multistep methods
    - additive operator splitting scheme (AOS)
    - ...
4. require a meeting to discuss with Hai
5. discuss with Tan.
6. coding

2 Expanding upon mcTangent

2.1 CNN Architecture

The paper describes using mcTangnet to approximate derivative calculation for simple Euler methods. Due to the finite difference schemes used to calculate the model-constrained losses, the network is encouraged to learn weights that resemble a convolutional neural network (CNN). Removing unnecessary connections by initializing a CNN layer as an input to the densely connected layer may accelerate the training process.

To begin testing this, training mcTangent with a CNN layer with a kernel of radius of 1 will approximate the central difference currently used in the model-constrained loss. This can be expanded to kernels with larger radii to learn finite differences that approximate the first order derivative with higher orders of accuracy.

2.2 Runge-Kutta Methods

Building on the current framework of approximating explicit methods, explicit Runge-Kutta methods are a generalization of the forward Euler method to higher orders. Training mcTangent to approximate the tangent manifolds at every intermediate time step will allow for each iteration to be completed with a single pass, rather than a series of calculations. Since Runge-Kutta methods call for multiple intermediate steps to be calculated, a deeper network architecture may be useful to capture this behavior.

A common such method is RK4 solves the initial value problem

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0$$

and takes the form

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$\begin{aligned} h &= t_{n+1} - t_n, \\ k_1 &= f(t_n, y_n), \\ k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{hk_1}{2}), \\ k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{hk_2}{2}), \\ k_4 &= f(t_n + h, y_n + hk_3) \end{aligned}$$

Some Runge-Kutta methods use adaptive time steps to manage truncation error at each iteration, capable of expanding or contracting the time step to efficiently reach the end state while remaining under the error threshold. This is done by comparing a method of order p to one of order $p-1$. These are selected so that they share intermediate steps, limiting any extra computation needed at each iteration. RK45 is a popular adaptive method with orders 4 and 5. It may be possible to train mcTangent to learn the derivatives necessary to approximate each iteration of RK45 as well as the adaptive timestep.

2.3 Implicit or Semi-implicit Methods

The small region of stability for explicit methods limits their usefulness over large time scales, as the small time steps required increases the number of steps required to reach the end state. Implicit methods are more stable and can thus provide better predictions on large time scales, despite the increase in computational cost. Since these costs arise from iteration or the solving of complex systems, training mcTangent to predict the derivatives needed in an implicit method can provide the stability with a lower cost with a trained network. However, the training process will remain expensive, as many implicit problems will need to be solved to provide the model-constrained loss.

2.4 Multistep Methods

Linear multistep methods are more accurate the single step equivalent (Euler method) with small enough time steps. McTangent can be trained to approximate the derivatives for each term in the S-order multistep method all at once, although doing so will mean mcTangent will no longer be agnostic to different time discretizations, as multistep methods such as Adams-Bashforth are dependent upon a stepsize in their calculations.

2.5 Additive Operator Splitting

Operator Splitting Methods simplify complex problems into simpler sub-problems, which can be solved more efficiently in sequence than the original problem all at once. Additive Operator Splitting takes this concept and instead solves the sub-problems in parallel, with each taking the same initial condition, rather than composing them. The independence of sub-problems, and their common initial condition, points to mcTangent as a possible way to solve all of the sub-problems in one pass.

2.6 JAX-Fluids

JAX-Fluids is a fluid dynamics solver written in JAX to facilitate the incorporation of ML models into fluid dynamics solvers.

3 Tests

3.1 Rerunning Forward Euler

Using `Generate_data` and`1st_forward_1_noise` to generate data for the 2D Burger equation and use `mcTangent` to approximate the calculation of the derivatives used in the Forward Euler method, instabilities were found in the predicted solution after training for 1e5 epochs with a learning rate of 1e-4. These results are shown in Figure 1.

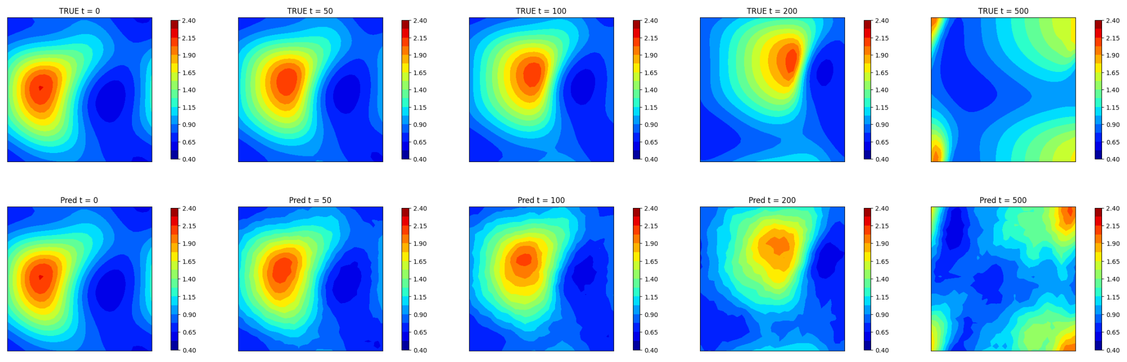

Figure 1: **Forward Euler.** Comparison of true and predicted solutions at various times. *Top Row*: Ground truth. *Bottom Row*: model-constrained network with noisy data

3.2 Reproduce Problem 1

`Generate_data` and `1st_forward_1_noise` were modified to produce the data and train the neural network as described in Problem 1 from Hai's paper. The network was trained for the unrandomized case with and without model constrained losses $(d200, 0\%, 1, 1, 0)/(d200, 0\%, 1, 1, 1e5)$ and the randomized case with and without model constrained losses $(d200, 2\%, 1, 1, 0)/(d200, 2\%, 1, 1, 1e5)$. These results are shown in Figure 2. The network trained with randomized training data and without model-constrained losses $(d200, 2\%, 1, 1, 0)$ diverged from the true solution after many time steps.

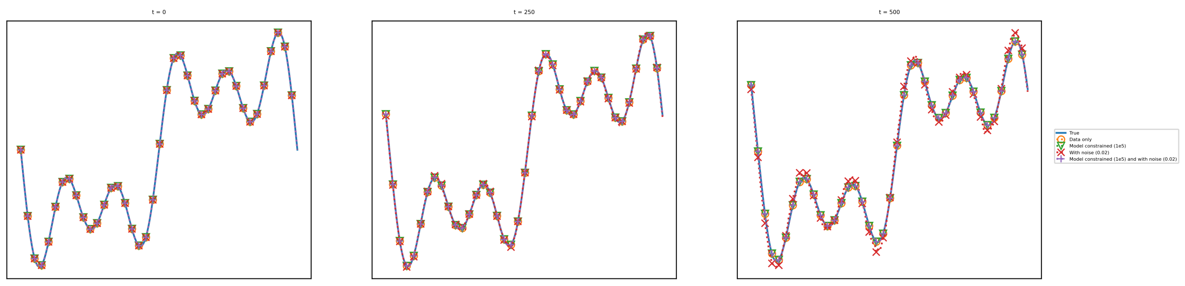

Figure 2: **1D Wave Transport.** Comparison of true and predicted solutions at various times.