Project Proposal

Hunter Johnson and Clayton Kramp

I. Description of Problem

For this project, we plan to develop a neural network solution to Ordinary Differential Equations and Partial Differential Equations. The neural network solution is to solve said equations efficiently and effectively. We will begin with equations that have analytical solutions, and the objective of the neural network solution is to beat common finite difference methods in accuracy and time of computation. Furthermore, we plan to test our neural network on equations without analytical solutions, and try to gain similar accuracy achieved by finite difference methods while improving time computation. We also plan to exploit the benefits of neural networks and test its accuracy of predicting values for future time, where a traditional approach would require precomputations of the past time points.

II. Challenges of Problem and Interests

Machine learning is a new topic for Clay and Hunter would like to extend his knowledge of the field. Not only is machine learning a massively growing field, but it is also an important topic in the upcoming years. Traditional fluid simulations and applications of finite difference often struggle with time of computation. As each element in a finite difference scheme depends on its neighbors, parallel computation becomes increasingly challenging. Thus a neural network approach could potentially make the problem significantly quicker to solve, and a better approach that does not have the common issues with finite difference methods such as stability, consistency, and convergence. Finite difference schemes, while may be accurate, must take small increments in time to be able to compute for its next time step. This makes future predictions, say for $t = 1000$ with $dt = 0.001$ extremely computationally expensive and inefficient in time. Neural networks implementations will be capable of estimating future predictions just as any other time step, creating drastic improvement in computation.

III. Approach and Rationale

We will begin by working with first order ODEs that have analytical solutions. This is to say that, if we begin with the following ODE:

$$\frac{dy}{dx} = x$$

We will have the following analytical solution:

$$y = x^2$$

The benefit of beginning with an ODE with an analytical solution, is that we can test our neural network's accuracy for future time steps very easily and quickly (ex, $\frac{dy}{dx} = x$ at $t = 2000$ we have $y = 2000^2$). This way, we can check for accuracy, computation time, and understand how memory hungry the network is to have accurate predictions.

We then move on to ODEs without analytical solutions. We will train our network with data provided with finite difference schemes, and test it with the finite difference scheme outputs. The part that we care the most about is the time it takes to compute the values. The benefit of the network is that it can compute for any time step equally, thus preventing the issue of finite difference schemes requiring previous time step data. This will also prevent the issue of finite difference schemes storing partial data to reach far in the future.

The real intriguing portion will be when we work with PDEs. With traditional methods, to compute for future time steps, we must evaluate the value at every spatial element, store it, and proceed.

This makes the computation for future time steps:

- Computationally expensive
- Spatially expensive
- Time costly

The goal of our implementation is to mitigate all three of the above restrictions. Finally, we would test our results with our network solution and finite difference solution, and (pray) that it produces similar results with exponentially less time.

Provided that our model works well until this point, we would like to apply it on to a real-world model. Currently, we are thinking about the Navier-Stokes Equation due to its high usage in real world fluid dynamics. We are currently thinking that applications of the network to deterministic models would provide better results, as nondeterministic models have huge error propagation and would not produce comparable results to its finite difference equivalent.

IV. Tools

Currently, we plan to use:

- Keras: which is a wrapper for TensorFlow
- autograd: python package for differentiation
- findiff: python package for finite differences
- matplotlib: python package for graphing
- (Maybe) Matlab: for graphing purposes

If time permits, we would also be interested in writing our own neural networks, and comparing the accuracy results with Keras results.

V. Evaluation Strategy

For equations with analytical solutions, we simply compare the network solutions with the exact solutions provided by the analytical equation. For equations without analytical solutions, we will be comparing our network solution with finite difference equation with high accuracy (aka, not Euler's method). Then, we would compare the results for large values of t, and compare the accuracy as well as the computation time. We will then examine errors between the two, and investigate if there is propagation in errors for large values of t. This will lead to investigation in amount of data necessary to train, and the number of neurons / depth of network we need to accurately model the problem. For 2D problems, we would have an error grid, and use heatmap on Matlab to see how the errors are distributed on a grid. Overall, we will be testing for accuracy and time cost of each implementation.

VI. Tentative Time Line

Time line with plans for each week:

1) Develop familiarity of deep neural network and develop model for first order ODE
2) Strong understanding of model and application with second order ODE, test with finite difference and acquire data
3) Apply model onto first order PDE, acquire data
4) Apply model onto second order PDE and develop method for breaking parameters into features
5) Comparing model to finite difference methods, acquiring data
6) Investigate other neural network architectures (residual, convolutional), begin application of model onto real world problem
7) Application of model onto real world problem (Navier-Stokes Problem)
8) Finalizing collection of data and writing report and presentation
9) Finishing up report and beamer presentation

VII. Description of Problem

For this project, we plan to develop a neural network solution to Ordinary Differential Equations and Partial Differential Equations. The neural network solution is to solve said equations efficiently and effectively. We will begin with equations that have analytical solutions, and the objective of the neural network solution is to beat common finite difference methods in accuracy and time of computation. Furthermore, we plan to test our neural network on equations without analytical solutions, and try to gain similar accuracy achieved by finite difference methods while improving time computation. We also plan to exploit the benefits of neural networks and test its accuracy of predicting values for future time, where a traditional approach would require precomputations of the past time points.

VIII. Challenges of Problem and Interests

Machine learning is a new topic for Clay and Hunter would like to extend his knowledge of the field. Not only is machine learning a massively growing field, but it is also an important topic in the upcoming years. Traditional fluid simulations and applications of finite difference often struggle with time of computation. As each element in a finite difference scheme depends on its neighbors, parallel computation becomes increasingly challenging. Thus a neural network approach could potentially make the problem significantly quicker to solve, and a better approach that does not have the common issues with finite difference methods such as stability, consistency, and convergence. Finite difference schemes, while may be accurate, must take small increments in time to be able to compute for its next time step. This makes future predictions, say for $t = 1000$ with $dt = 0.001$ extremely computationally expensive and inefficient in time. Neural networks implementations will be capable of estimating future predictions just as any other time step, creating drastic improvement in computation.

IX. Approach and Rationale

We will begin by working with first order ODEs that have analytical solutions. This is to say that, if we begin with the following ODE:

$$\frac{dy}{dx} = x$$

We will have the following analytical solution:

$$y = x^2$$

The benefit of beginning with an ODE with an analytical solution, is that we can test our neural network's accuracy for future time steps very easily and quickly (ex, $\frac{dy}{dx} = x$ at $t = 2000$ we have $y = 2000^2$). This way, we can check for accuracy, computation time, and understand how memory hungry the network is to have accurate predictions.

We then move on to ODEs without analytical solutions. We will train our network with data provided with finite difference schemes, and test it with the finite difference scheme outputs. The part that we care the most about is the time it takes to compute the values. The benefit of the network is that it can compute for any time step equally, thus preventing the issue of finite difference schemes requiring previous time step data. This will also prevent the issue of finite difference schemes storing partial data to reach far in the future.

The real intriguing portion will be when we work with PDEs. With traditional methods, to compute for future time steps, we must evaluate the value at every spatial element, store it, and proceed.

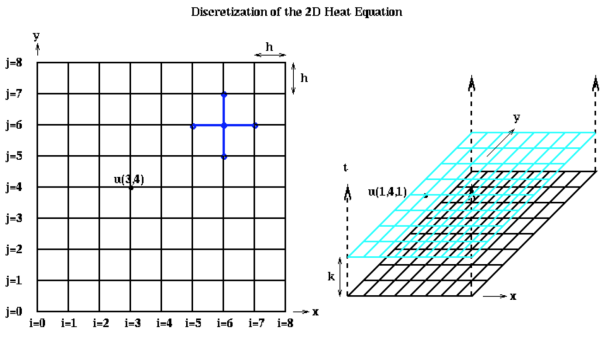

Fig. 1. Partial Differential Equation Solve with Finite Difference

This makes the computation for future time steps:

- Computationally expensive
- Spatially expensive
- Time costly

The goal of our implementation is to mitigate all three of the above restrictions. Finally, we would test our results with our network solution and finite difference solution, and (pray) that it produces similar results with exponentially less time.

Provided that our model works well until this point, we would like to apply it on to a real-world model. Currently, we are thinking about the Navier-Stokes Equation due to its high usage in real world fluid dynamics. We are currently thinking that applications of the network to deterministic models would provide better results, as nondeterministic models have huge error propagation and would not produce comparable results to its finite difference equivalent.

X. Tools

Currently, we plan to use:

- Keras: which is a wrapper for TensorFlow
- autograd: python package for differentiation
- findiff: python package for finite differences
- matplotlib: python package for graphing
- (Maybe) Matlab: for graphing purposes

If time permits, we would also be interested in writing our own neural networks, and comparing the accuracy results with Keras results.

XI. Evaluation Strategy

For equations with analytical solutions, we simply compare the network solutions with the exact solutions provided by the analytical equation. For equations without analytical solutions, we will be comparing our network solution with finite difference equation with high accuracy (aka, not Euler's method). Then, we would compare the results for large values of t, and compare the accuracy as well as the computation time. We will then examine errors between the two, and investigate if there is propagation in errors for large values of t. This will lead to investigation in amount of data necessary to train, and the number of neurons / depth of network we need to accurately model the problem. For 2D problems, we would have an error grid, and use heatmap on Matlab to see how the errors are distributed on a grid. Overall, we will be testing for accuracy and time cost of each implementation.

XII. Tentative Time Line

Time line with plans for each week:

1) Develop familiarity of deep neural network and develop model for first order ODE
2) Strong understanding of model and application with second order ODE, test with finite difference and acquire data
3) Apply model onto first order PDE, acquire data
4) Apply model onto second order PDE and develop method for breaking parameters into features
5) Comparing model to finite difference methods, acquiring data
6) Investigate other neural network architectures (residual, convolutional), begin application of model onto real world problem
7) Application of model onto real world problem (Navier-Stokes Problem)
8) Finalizing collection of data and writing report and presentation
9) Finishing up report and beamer presentation