Neural Network assisted numerical noise reduction in central difference schemes for solving hydro-dynamic systems of partial differential equations

Gaute Holen
Department of Physics
project github repo

June 9, 2023

Abstract

A neural network is merged with a finite difference scheme solver and trained to set diffusive constants for artificial viscocity used to reduce numerical noise and unstable oscillations a hydro-dynamic system of partial differential equations. The diffusive constants are traditionally constant in time, and their values will determine the characteristics of shocks and discontinuities while having an overall diffusive effect. Here, rather than using the neural network to approximate the solution to the PDE system, it is assisting the finite difference scheme. The goal of the network is to update the diffusive constants every timestep to have just enough diffusion to tame unstable oscillations while having a minimal diffusive effect on the overall system. The network is trained and testeed on the Sod shock problem, where the network performs better than diffusive constants that are constant in time, but worse than conservative schemes such as the Lax-Wendroff or Roe first order.

Contents

1 Introduction

In the age of computers everywhere, supercomputers and machine learning we are now looking for new ways to implement neural networks to make better use of the computational resources available. Simulations and numerical solutions to equations is a core part of the field of plasma physics. This is because it is very difficult to collect data on the large scales required to understand space plasma that impact the sun-earth system. As a result, simulations are used as they produce superior data. However, a simulation is nothing more than an approximation, and deciding the importance and necessary degree of approximation for different aspects of the physics in a system proves to be a major challenge. In order to simulate the universe to exact precision, we would need a simulation larger than the universe and so on. Yet, the ultimate goal is to merge small-scale and large-scale phenomena, and simulate both simultaneously. As such, the field is always looking to exploit computational resources with maximum efficiency, and deploy any emerging new technology to help achieve better approximations.

In plasma physics, we are particularly interested in solving a continuity equation, momentum equation and energy equation (Hydrodynamics) also known as the Euler equations, where we implement electromagnetic forces (Magnetohydrodynamics). This is typically done with Particle in cell [1] or fluid simulations employing various numerical schemes. From the universal approximation theory [2, 3], we know that we can approximate any continuous function with a feed forward neural network of at least one hidden layer and a sigmoidal activation function. Neural networks have been used to approximate the to partial differential equations including hydrodynamics [4], but per now the state of the art within the field is not neural networks. One reason for this is that we already have many ways to solve our equations numerically, and unless the neural network provides better efficiency or scalability it's not helpful. Neural networks are good at complicated many-dimensional non-linear problems, and solving something like

$$\frac{\partial h(x,t)}{\partial t} + \nabla \cdot h(x,t) = 0 \tag{1}$$

is not really such a problem. Instead, in this investigation we propose to solve the easy part of the physics with traditional methods, and leave the non-linear tricky parts to the neural network. As such, we get the efficiency and accuracy of solving the trivial parts of the problem with well known and optimized methods, and can leave the complicated part of the problem to the neural network. Essentially, we propose merging a classic finite difference scheme with some quantity that can be adjusted by the network for optimization or understanding non-linear behaviour. Assume we have some physical quantity $u^n(t,x)$ which varies in time and space, which we have discretized and where the next timestep u^{n+1} is approximated by

$$u^{n+1}(x,t) = f(u^n(x,t), c_{pred}) \tag{2}$$

Where f is some finite difference scheme that takes the current state of the system $u^n(x,t)$ but also takes some parameter c_{pred}that has been predicted by a neural network whose input is $u(x,t)$. Expaning it we get

$$u^{n+1}(x,t) = f(u^n(x,t), N(W, X(u(x,t))) \tag{3}$$

Where N is a neural network with weights W and input X which is a function of $u(x,t)$ such that $N(W, X(u(x,t))) = c_{pred}$. The cost function depends on how well our predicted timestep u^{n+1} compares with an analytical solution or perhaps the solution of a more accurate numerical scheme y^{n+1}_{exact} such that

$$C(u^{n+1}, y^{n+1}_{exact}) = C(f(u^n(x,t), N(W, X(u(x,t))), y^{n+1}_{exact}) \tag{4}$$

It now becomes apparent that finding the derivative of the cost function is not trivial, something that proves to be a major challenge in this investigation. Another way to understand this setup is that we have a neural network with a finite difference scheme on the output layer, which is complicated, particularly with a coupled system of partial differential equations. Keeping in mind that with a standard neural network the output is the final prediction to evaluate in the cost function, it becomes clear that standard approaches no longer work for this type of hybrid solver network setup. In some cases the universal approximation theorem still holds for extra functions or steps on the output layer[5]. However it is unclear whether it holds in our case.

Reminding ourselves that the motivation for this type of new setup, however unconventional, is to get the best of both the well understood opitmized numerical schemes and neural networks' ability to understand non-linearities. Therefore, we will try to implement a working model based on solving the 1D Euler equations for the well known Sod Shock problem [6]. Many phenomena in plasma physics are understood through linearization, however there are many non-linear phenomena that are not well understood. Although we don't directly tackle any new non-linear physics in this investigation, as we are implementing non-linear understanding of the physics it might be a step in that direction. As such, this investigation explores a new way to use neural networks to assist the simpler numerical scheme. Additionally, we hope that the outputs of the neural network can help us understand something about the non-linearity of the system.

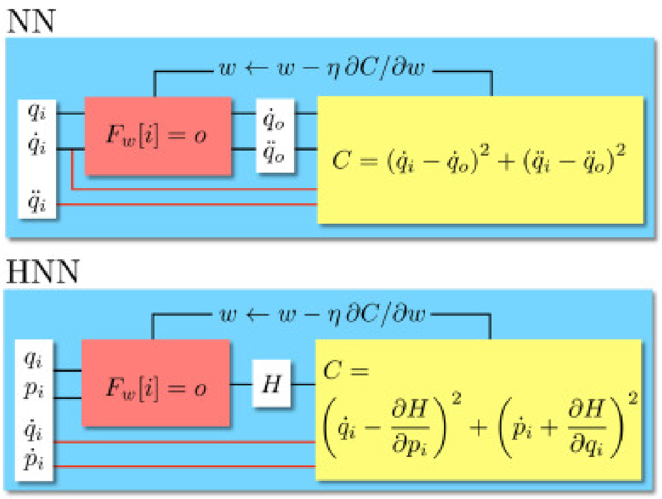
NN
$w \leftarrow w - \eta\,\partial C/\partial w$
q_i
$\dot{q}_i$
$\ddot{q}_i$
$F_w[i] = o$
$\dot{q}_o$
$\ddot{q}_o$
$C = (\dot{q}_i - \dot{q}_o)^2 + (\ddot{q}_i - \ddot{q}_o)^2$
HNN
$w \leftarrow w - \eta\,\partial C/\partial w$
q_i
p_i
$\dot{q}_i$
$\dot{p}_i$
$F_w[i] = o$
H
$C = \left(\dot{q}_i - \frac{\partial H}{\partial p_i}\right)^2 + \left(\dot{p}_i + \frac{\partial H}{\partial q_i}\right)^2$

Figure 1: Hamiltonian neural network which computes the Hamiltonian from which the desired quantities are calculated. As such there is an extra output layer with some derivatives being calculated. [7]

It is currently unclear whether this type of setup has been tried. However, the Hamiltonian neural networks use a similar approach where the network does not approach the specific quantity we are interested in, but approaches the Hamiltonian of the system from which the specific quantities are calculated. For dynamic system it has been shown to perform a lot better than computing specific quantities directly[7]. The difference from a standard neural network and a Hamiltonian neural network is seen in Figure 1 [7]. Our model has a similar structure which can be seen in Figure 6 and will be described in more detail later. As the Hamiltonian proves to be better than directly approximating the solution, we hope that this also indicates that our proposed model will be fruitful.

First, we will give a short introduction to the physics and some relevant existing numerical methods, then we will look at the model and its design and implementation challenges. Lastly, we will look at and discuss the results for the Sod Shock test. We have used the Sod Shock test [6] because it has a discontinuity, sharp gradients and shockwaves which is a great way to test numerical codes in fluid dynamics problems.

2 Background Theory

Here, you'll find some very brief background info on the phsyics and numerical schemes used in this project. However, the focus here is on the neural network, not on the numerical schemes, so this is kept extremely brief and oversimplified. A lot of the info here is part of the course AST5110. [1]

2.1 Hydro-dynamic PDE model

In plasma physics and solar physics, Hydro-dynamic PDE models are used to simulate each species in the plasma. To keep things simple, we will move forward with a simplified version of Euler's equations [8] for mass, momentum and energy such that

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{5}$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla \left(P_g\right) \tag{6}$$

$$\frac{\partial e}{\partial t} = -\nabla \cdot e\mathbf{u} - P_g \nabla \cdot \mathbf{u} \tag{7}$$

Where ρ is the density, $\mathbf{u}$ is the velocity, Pg is the gas pressure and e is the energy. In our case we consider only 1 dimension such that

$$\nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = \frac{\partial \left(\rho u^2\right)}{\partial x} \tag{8}$$

Additionally there's an equation of state relating the pressure to the energy of the system

$$Pg = (\gamma - 1)(e - \rho \mathbf{u}^2) \tag{9}$$

For more useful implementations of these equations things such as electromagnetic force and collisions between multiple species are added. But for now we stick to this simple version.

2.2 Numerical schemes

Partial differential equations on the form

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0 \tag{10}$$

Can be discretized assuming $\frac{\partial u}{\partial t} = \frac{u_j^{n+1} - u_j^n}{dt}$ and $\frac{\partial u}{\partial x} = \frac{u_{j+1}^n - u_{j-1}^n}{2dx}$. Solving for the next time step u_j^{n+1} we obtain

$$u_j^{n+1} = u_j^n - \frac{dt}{2dx}(u_{j+1}^n - u_{j-1}^n) \tag{11}$$

Which is the central difference scheme. There is a lot to be said about the different schemes, but for simplicity we will stick to the central difference scheme. In Figure 2 we see the solution to Euler's equation as presented in the previous section with a simple central difference scheme for the Sod shock tube problem [6]. Note the oscillations due to to numerical noise.

[1] https://www.uio.no/studier/emner/matnat/astro/AST5110/

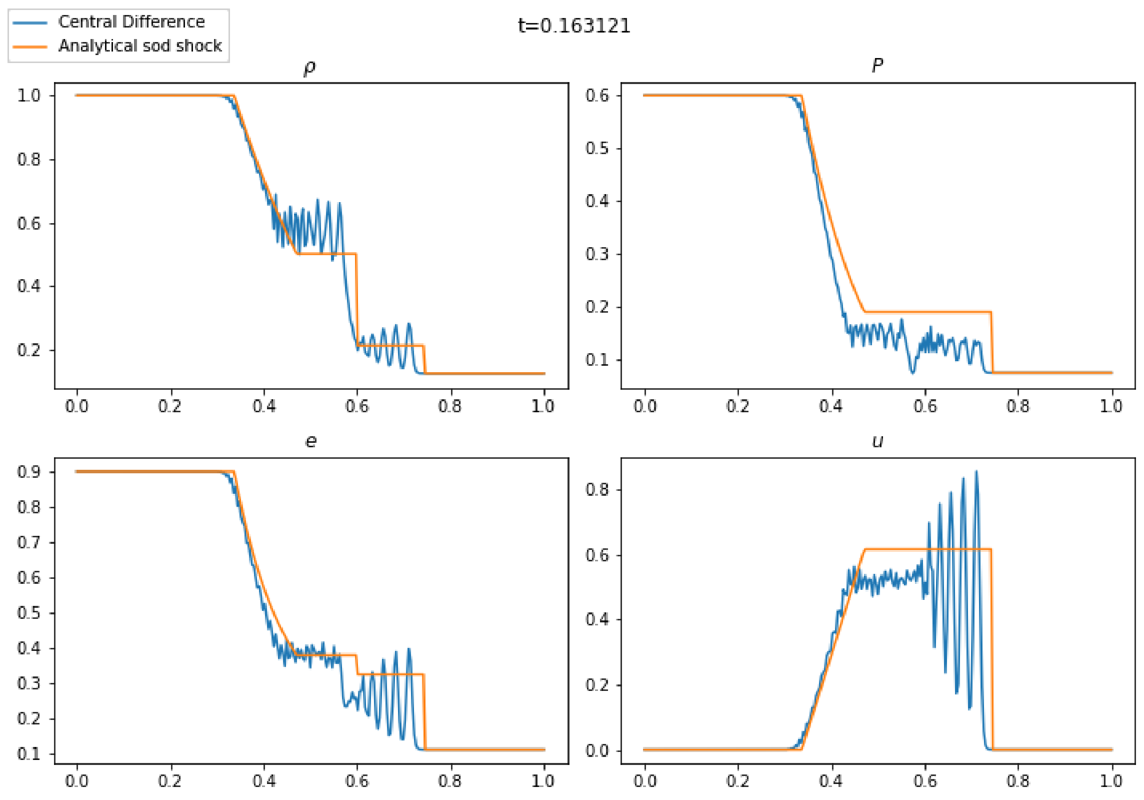

Figure 2: Sod test tube with central difference scheme and analytical solution for 256 points.

2.3 Artificial Viscosity

One way to combat the numerical noise is to introduce artificial viscosity. Although artificial, it can physically be interpreted as any kind of damping or friction that prevents a system from oscillating indefinitely. Artificial viscosity was first introduced by Richtmayer and Von Neumann [9] in the 1950s by adding q to the pressure term where

$$q_{RvN} = -\rho c_q \Delta x^2 \frac{\partial^2 u}{\partial x^2} \tag{12}$$

Where c_q is some constant typically set to about 2. Later improvements have been made summed up by L. G. Margolin and N. M. Lloyd–Ronnin [10] where

$$q = -\rho \left(c_q \Delta x^2 \frac{\partial^2 u}{\partial x^2} + c_L \Delta x \frac{\partial u}{\partial x} \right) \tag{13}$$

Where c_q and c_L are some constants. Here, the characteristics of the shock will be determined largely by c_q and c_L. However, it appears to largely be determined by trial and error, something that is great for a neural network to determine.

Lastly, as the pressure term is not present in the continuity equation, a similar diffusion term is added where

$$q_d = D \frac{\partial \rho u}{\partial x} \tag{14}$$

Where $D = c_D \Delta x \rho$ and c_D is some unitless constant. Without the diffusive term in the continuity equation we get oscillations in the density whereas the shock front is largely under control. Finally, we obtain the equations

$$\frac{\partial \rho}{\partial t} + \nabla \cdot ((\rho + q_d)\mathbf{u}) = \mathbf{0} \tag{15}$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla (P_g + q) \tag{16}$$

$$\frac{\partial e}{\partial t} = -\nabla \cdot e\mathbf{u} - (P_g + q)\nabla \cdot \mathbf{u} \tag{17}$$

Now we have the set of equations we will use with the neural network, where the constants c_q, c_L and c_D will be set by the network for every time step. In Figure 3 we see the improvements of the sod shock test with constant c_q, c_L and c_D set constant for the whole run.

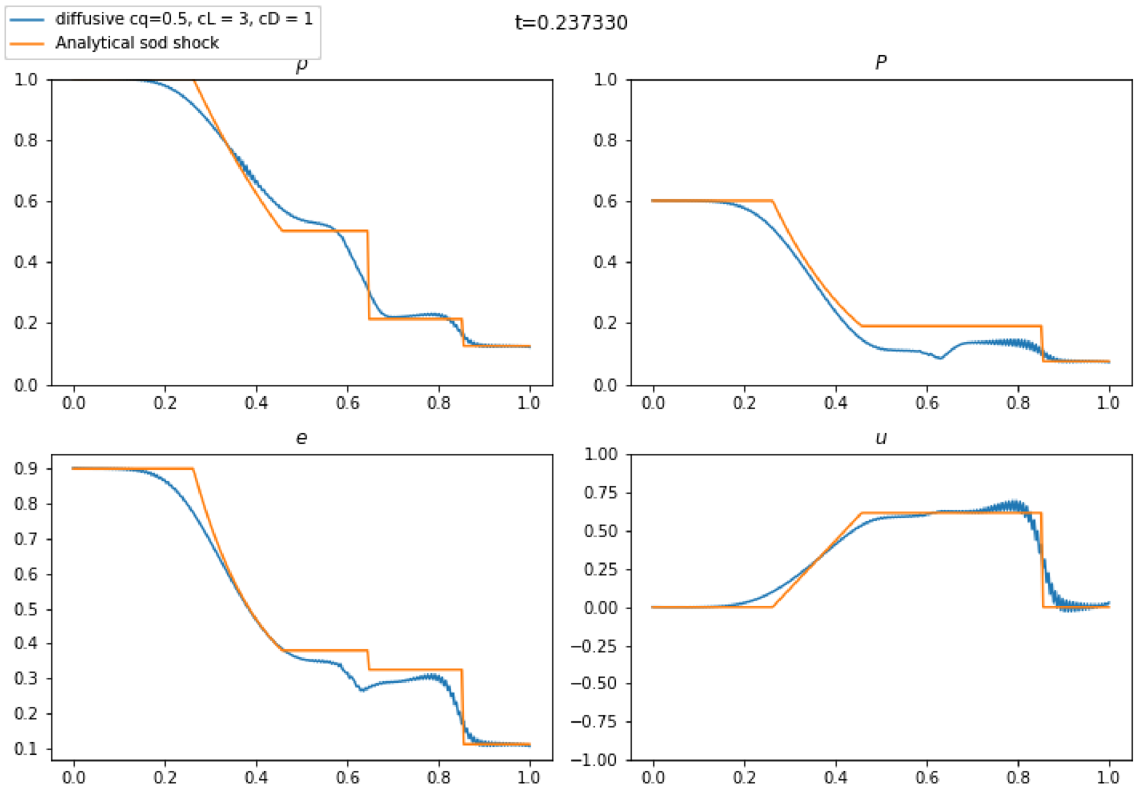

Figure 3: Sod test tube with central difference scheme with artificial diffusion and analytical solution for 256 points.

2.4 Conservative schemes

If our equations are conservation laws we can use conservative schemes. Here, we compute the flux between the cells in space, and say that our change of U in time must be a balance of the interface flux between the cells at $j \pm 1/2$ and creation or destruction of U (source term S) where the flux F is defined as:

$$\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} = S \tag{18}$$

Where source term S is often set to zero (if no U is created or destroyed). From equations 5, 7 and 6 we obtain

$$\vec{F} = \begin{pmatrix} \rho \mathbf{u} \\ \rho \mathbf{u}^2 + P_g \\ -e\mathbf{u} - P_g \mathbf{u} \end{pmatrix}, \vec{U} = \begin{pmatrix} \rho \mathbf{u} \\ \rho \\ e \end{pmatrix} \tag{19}$$

And we can move forward in time with the scheme

$$U^{n+1} = U^n - \frac{dt}{dx} dF \tag{20}$$

Here, dF denotes the interface fluxes and can be calculated in many ways depending on which scheme is used. For our case, we will consider the Lax-Wendroff scheme such that

$$dF_j = F_{j+1/2} - F_{j-1/2} \tag{21}$$

To remain concise we will stop there.

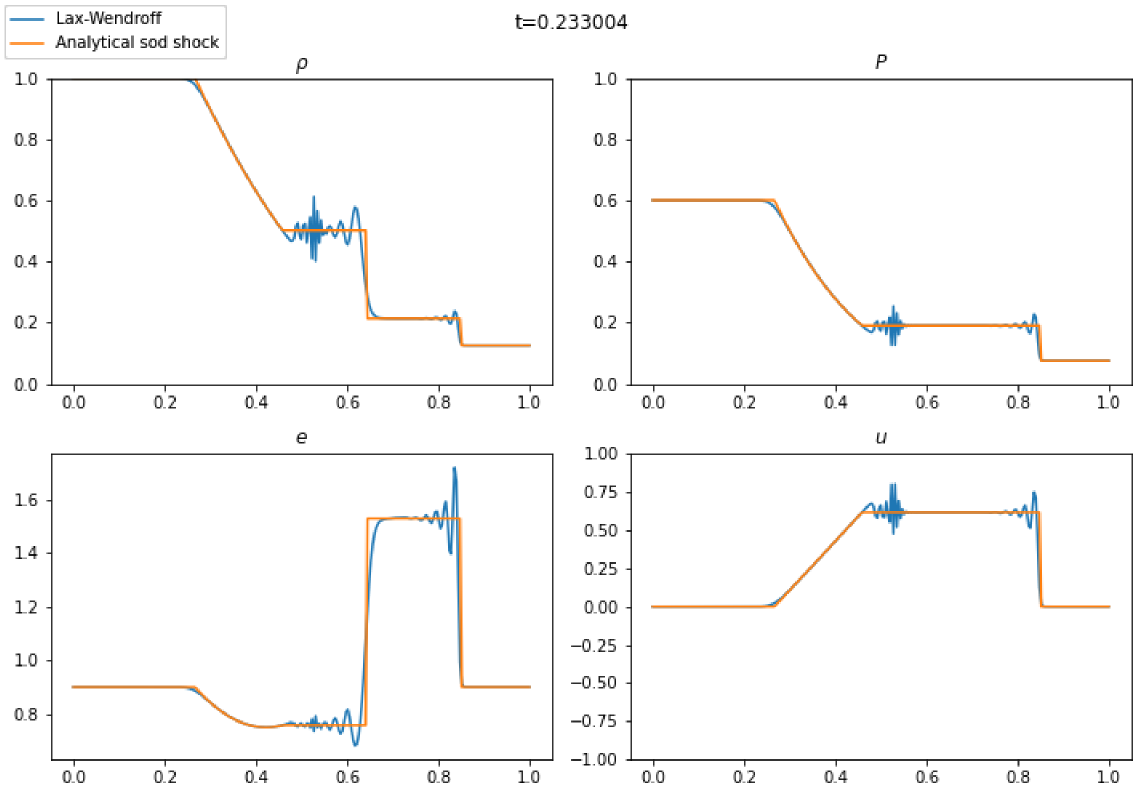

Figure 4: Sod test tube with Lax-Wendroff scheme and analytical solution for 256 points

The Roe shceme (Riemann solver), breaks the problem down by stating that

$$\frac{\partial U}{\partial t} + A(\boldsymbol{U})\frac{\partial U}{\partial x} = 0 \tag{22}$$

Where A is the Jacobian Matrix of the flux vector $F(U)$. The Roe solver finds the matrix $A(U_j, U_{j+1})$ that is assumed constant between the two cells. This method involves matrix inversions and operations on every pair of cells in space, and is not very efficient computationally.

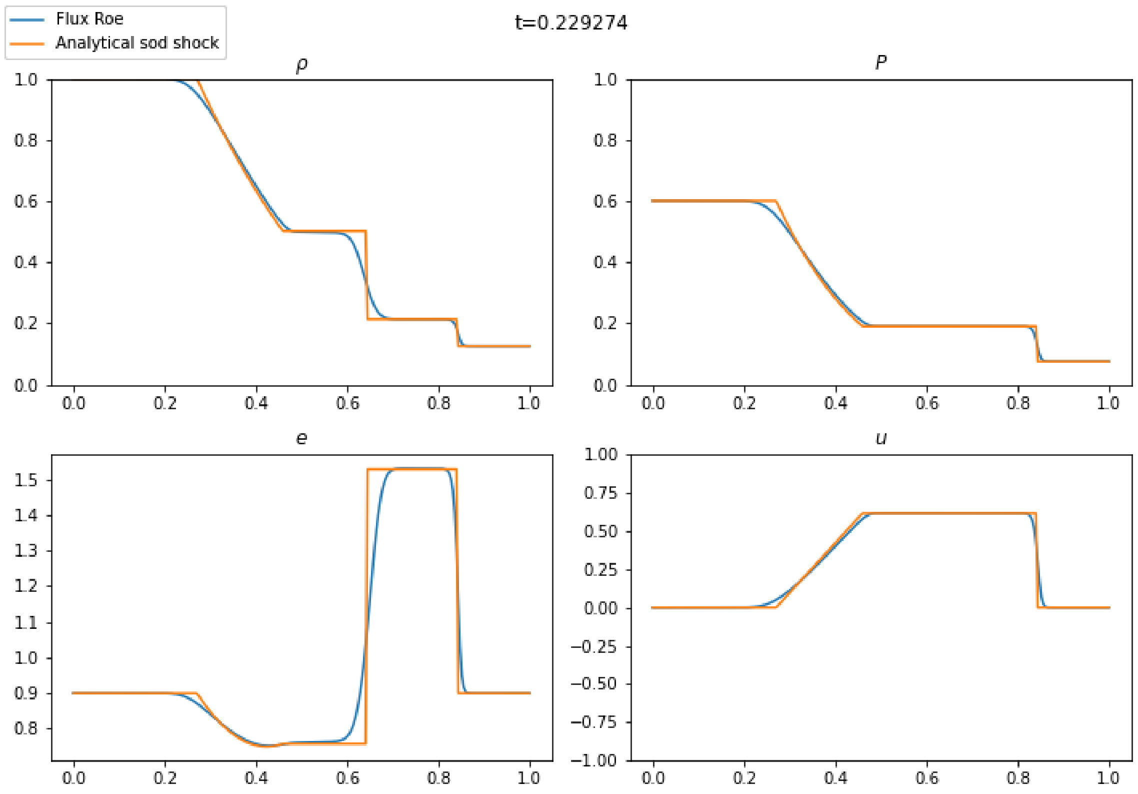

Figure 5: Sod test tube with Roe Riemann solver and analytical solution for 256 points

3 Design and implementation of the model

The goal of the model is to predict the diffusive constants c_q, c_L and c_D in the Equations 13 and 14 that result in the best approximation for the hydro-dynamic equations 6, 5 and 7 using the parameters $\mathbf{u}$, ρ, e and P_g respectively velocity, density, energy and gas pressure.

The outputs will be 3 nodes (one for each diffusive constant), and the inputs are the velocity, density, energy and gas pressure all combined into one column vector X. If we have 256 points in space, then X has length of 1024. The general idea for the hidden layers is reducing the size of the hidden layer gradually from 1024 to 3. Also, we want the outputs to always be positive, otherwise we get unphysical behaviour. In Table 1 the dimensions and activation of the neural network for 256 points in space in shown.

Table 1: Sigmoid model: The input layer, hidden layers, output layer and integration layer with 256 spacial points

Layer	Nodes	Activation/Scheme
Input	1024	Sigmoid
Hidden	256	Sigmoid
Hidden	85	Sigmoid
Hidden	30	Sigmoid
Hidden	9	Sigmoid
Output	3	Abs(X)
Integration	1024	Central difference

Table 2: Tanh model: The input layer, hidden layers, output layer and integration layer with 256 spacial points

Layer	Nodes	Activation/Scheme
Input	1024	Tanh
Hidden	256	Tanh
Hidden	85	Tanh
Hidden	30	Tanh
Hidden	9	Tanh
Output	3	Abs(X)
Integration	1024	Central difference

The model is based on a feedforward neural network with backpropagation to update the weights. However, the target is the analytical solution to the problem, whereas the output of the network is the diffusion constants. In order to evaluate how well the model's prediction is, we need to perform a timestep with the predicted diffusion constants and compare the resulting velocity, density, energy and gas pressure to the target.

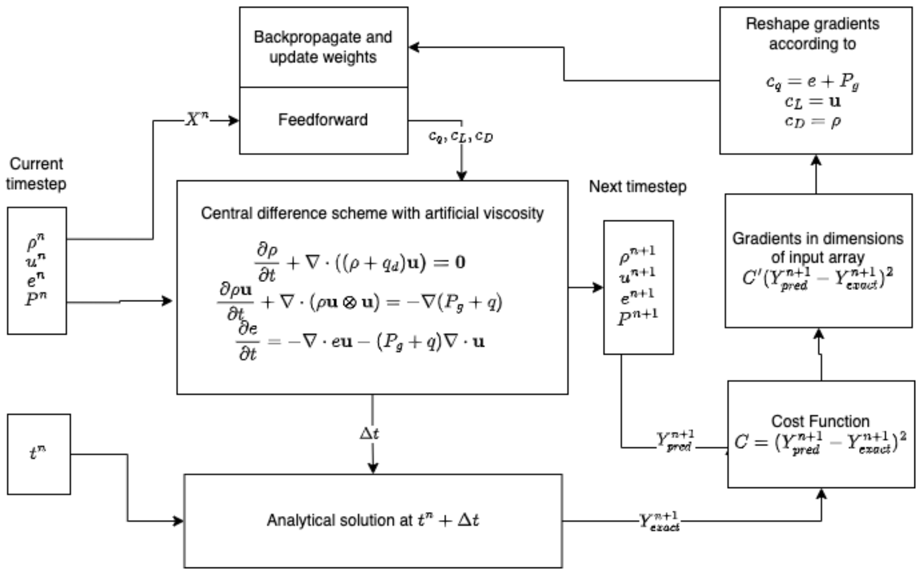
Backpropagate and update weights
Feedforward
X^n
c_q, c_L, c_D
Reshape gradients according to
$c_q = e + P_g$
$c_L = \mathbf{u}$
$c_D = \rho$
Current timestep
ρ^n u^n e^n P^n
Central difference scheme with artificial viscosity
$\frac{\partial \rho}{\partial t} + \nabla \cdot ((\rho + q_d)\mathbf{u}) = \mathbf{0}$
$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla(P_g + q)$
$\frac{\partial e}{\partial t} = -\nabla \cdot e\mathbf{u} - (P_g + q)\nabla \cdot \mathbf{u}$
Next timestep
ρ^{n+1} u^{n+1} e^{n+1} P^{n+1}
Gradients in dimensions of input array
$C'(Y^{n+1}_{pred} - Y^{n+1}_{exact})^2$
Δt
t^n
Y^{n+1}_{pred}
Cost Function
$C = (Y^{n+1}_{pred} - Y^{n+1}_{exact})^2$
Analytical solution at $t^n + \Delta t$
Y^{n+1}_{exact}

Figure 6: Overview of how the network and central difference scheme is connected. Here the supersctipt n the values at some timestep n, and the superscript $n + 1$ denotes the updated values at the next timestep.

The main challenge of this type of model is that we want to fit an integration step in the middle of the model, which employs a numerical method for solving a system of partial differential equations. In some ways we have an "implicit" output step that also becomes part of the backpropagation as seen in Figure 6 the Typical automatic differentiation algorithms are particularly tricky to integrate with this type of "hidden" or implicit output step, so this is a major area of improvement.

3.1 Backpropagation

As mentioned already, fitting the timestepping into the backpropagation proves to be challenging. The gradient of the costfunciton can't be automatically differentiated (or differentiated at all?) as there is an integration step inside it. The backpropagation is initialized by taking the derivative of the costfunction C evaluated at the prediction timestep y_{pred} and target y, and the derivative of the output function O evauluated at $O(W_n \cdot C'(y_{pred}, y))$ where W_n is the weights at the output layer. The problem here, is that the dimensions of W_n is 3 and the dimensions of y_{pred} and y are 1024. This is because the cost function evaluates after the timestep.

As a result we need to reduce the dimensionality of $C'(y_{pred}, y))$, which now gives us the gradient of the cost function in 1024 points, and we need it in 3 points. We know that the input is made up of density, velocity, energy and gas pressure all combined together. Thus, we can assume that $C'(y_{pred}, y))$ can be divided into the effect the three diffusive constants c_q, c_L and c_D have on the gradients in $C'(y_{pred}, y))$. If we make some assumptions about how each of the density, velocity, energy and gas pressure affect the diffusive constants, then we can help the network by reducing the dimensionality of $C'(y_{pred}, y))$ according to Table 3.

Table 3: The assumed relation between $C'(y_{pred}, y))$ and the diffusive constants. Here, $C'(y_{pred}, y))$ is sliced in four according to how it combines density, velocity, energy and gas pressure in the input layer.

Diffusive constant	Relation to $C'(y_{pred}, y))$
c_q	$e + Pg$
c_L	$\mathbf{u}$
c_D	ρ

All the parameters are interconnected, and the assumption made in Table 3 is a major area of improvement and a gross simplification. The assumption is based on playing with the values of the diffusive constants in a solver to try to see which diffusive constant was dominant in each parameter. That being said, this gives the network enough to learn something about how to set the diffusive constants.

However, evaluating $O(W_n \cdot C'(y_{pred}, y))$ does not seem to help the network learn as it only adds to the values of the diffusive constants and doesn't reduce them. As we have already strayed from the standard approach, there is no reason why including the output function gradient is a must if it breaks the learning. Using only the gradient from the cost function which includes a timestep is sufficient to for the network to learn. The final procedure for the output layer of the backpropagation then becomes:

1. Predict the values of c_q, c_L and c_D
2. Perform a timestep with the predicted constants
3. Find the gradient of the cost function evaluated at the new timestep and the target. This is the same as $C'(y_{pred}, y)$ with 1024 values
4. Split $C'(y_{pred}, y)$ into subarrays of 256 values, assumed to correspond to the physical paramaters density, velocity, energy and gas pressure
5. Reshape $C'(y_{pred}, y)$ into 3 values corresponding to the mean of the values like in Table 3
6. Set the delta matrix as the new 3 values
7. Move on with standard backpropagation

Overall the challenge here is to sufficiently tell the network something about how the gradient of the weights, output function and cost functions work with the timestepping and reduction of dimensionality. While the approach used here is not perfect, or even good, it is enough for the network to learn something.

3.2 Activation functions

The purpose of the activation function is to introduce non-linearity to the network. In our case, we assume a non-linear relationship between the input and outputs, especially with the integration step. The purpose of the output acitivation function is to limit the range of values exiting the network. and To prevent unphysical behaviour we have chosen the absolute value, which simply makes sure that the outputs are always positive or zero. Since it is not entirely clear which activation function to use for the hidden layers for this network, we have chosen two very common ones (which also seemed to work well): The sigmoid and the tanh.

The Sigmoid has a range of 0 to 1, which means that there can only be a positive or zero contribution between layers. The sigmoid is defined as

$$f(x) = \frac{1}{1 + e^{-x}} \tag{23}$$

The range of the sigmoid makes it ideal for probability problems, which our problem might not fit. Also, it might run into the vanishing gradients problem as the gradients get very small at larger values.

The tanh is similar to the sigmoid, however it has a range of -1 to 1, which means that we get both positive, negative and neutral contributions from the nodes. In our case, this leads to more erratic and rapidly changing behaviour. The tanh is deined as:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{24}$$

Tanh also suffers from the vanishing gradient problems, as it has very small gradients outside of about -5 to 5.

The identity activation was also tried, but it gave no meaningful output and time was not spent investigating why outside of changing the learning rate.

3.3 Cost function

The cost function is the function used to evaluate how good our network did, so it can update it's weights accordingly. Typically, the cost function takes the output of the network y_{pred} and compares it to some target y. In our case we need to also add the integration step between the output and the cost function. The procedure is as follows:

1. Get the predicted diffusive constants from the network c_q, c_L and c_D
2. Move forward in time and get the predicted y_{pred}
3. Find the difference between y_{pred} and y with some cost function $C(y_{pred}, y)$

In our case, we are using OLS to find the error, such that:

$$C(x, y) = \frac{1}{N}\sum_{i=0}^{i=N}(x_i - y_i)^2 \tag{25}$$

Where x is the predicted timestep, y is the target, and N is the number of points. In our case the target is the analytical solution to the sod test problem.

To accurately gauge the error of our model we start at the initial condition and run the numerical solver with the predicted diffusive constants over all the timesteps in our data. This takes some time, so the error is only calculated every 10 epochs.

3.4 Target, Timestepping and lookahead

For the simplest configuration of the network, the input and targets are both based on the analytical solution of the sod test tube. Here, the input X will be the analytical solution X_{exact} at time n, and the target y will be the analytical solution at time $n+1$. As such we get:

$$X = X_{exact}^{n}, y = X_{exact}^{n+1}$$

The timestep is constant in time, and Δt is set to some value lower than the CFL stability condition for the solver. For the central difference with 256 points this is around $3E-3$, and we use $2E-3$ for some extra caution.

Two major issues with this approach are:

1. Numerical noise and oscillations are built up over time, and won't be easily be spotted after just one timestep.
2. Starting from an exact solution is only valid for the first step. All consecutive steps will start from the predicted previous timestep and not the previous analytical solution

The first of these issues are attempted to be handled by introducing lookahead to the model. That is, the model does not just move one timestep with the predicted diffusive constats, it moves m timesteps. As such, the input and target will be:

$$X = X_{exact}^{n}, y = X_{exact}^{n+m}$$

One benefit from this lookahead is that it allows the model to see if any diffusive constants cause an accumulation of error over many timesteps. However, when the model is actually used with the solver it changes the diffusive constants every timestep, so it puts some constraints on the training that are not present in the actual use case. In Figure 7 a simple diagram shows how the lookahead is implemented.

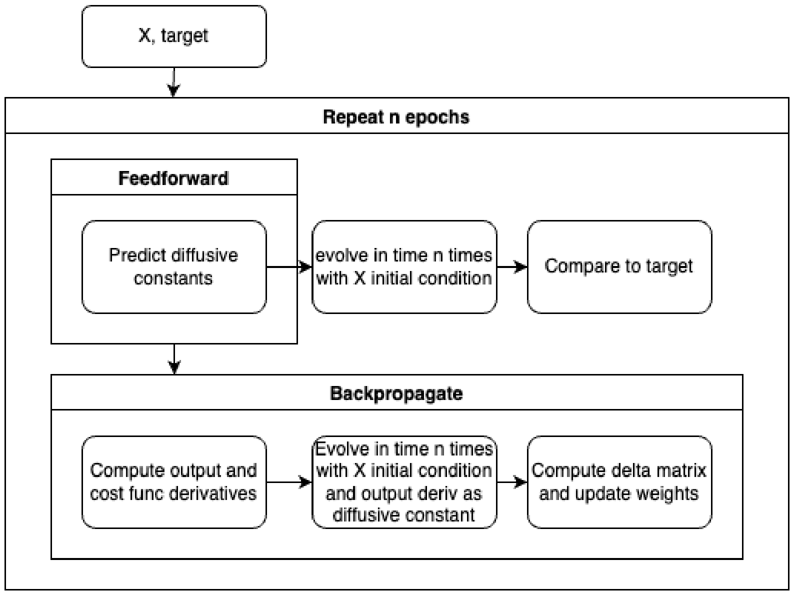
X, target
Repeat n epochs
Feedforward
Predict diffusive constants
evolve in time n times with X initial condition
Compare to target
Backpropagate
Compute output and cost func derivatives
Evolve in time n times with X initial condition and output deriv as diffusive constant
Compute delta matrix and update weights

Figure 7: Diagram of how the integration step fits in the neural network with lookahead

Table 4: The effect of lookahead. For sigmoid with 20 epochs, 3 iterations and initial learning rate 9E-2. For tanh 30 epochs, 3 iterations and initial learning rate 3E-2

Lookahead	Sigmoid OLS	tanh OLS	Total timesteps
1	1.59 E-3	1.60 E-3	3.0 E4
5	1.59 E-3	1.58 E-3	1.5 E5
10	1.59 E-3	1.59 E-3	3.0 E5
20	1.58 E-3	1.58 E-3	6.0 E5
50	1.56 E-3	1.51 E-3	1.5 E6
100	1.52 E-3	1.51 E-3	3.0 E6

So how does the lookahead perform? In Table 4 the increased lookahead does give better results. However, the total timesteps performed also increases by a factor of m (the lookahead). To see the effect this has on performance we need to know what amount of time is spent doing the timestepping. In Table 5 it is clear that for high lookahead the timestepping becomes the dominating part of the model training.

Table 5: Profiling of training performed on 256 points for 1500 timesteps for different lookahead. The timesteps is the total amount of timestep

Training parameters		Training time			Timestep		
Epochs	Lookahead	Total	FF cum	BP cum	nsteps	cumtime	per call
10	1	3.877	1.944	1.904	1.5E4	0.209	1.3E-5
10	50	14.906	2.217	11.903	7.5E5	9.788	1.3E-5

3.5 Iterative feedback

As mentioned already, using the exact solution as the starting point is only valid for one timestep. One approach to combat this is to iteratively feed back it's outputs as the new inputs. Let's say we start out by training according to

$$X = X^n_{exact}, y = X^{n+1}_{exact}$$

Then, we run the solver with the predicted constants and get an output y_{pred}. This output will have all the problems associated with the numerical noise and oscillations. Now, we use y_{pred} as the starting point so that the network learns how to take the noisy input produced by the solver and help it better approach the exact solution. This looks like:

$$X = y^n_{pred}, y = X^{n+1}_{exact}$$

This can be repeated over and over, where the idea is that as the model gets better at correcting for the numerical noise, the inputs must be updated to the less noisy outputs so the model doesn't do too much or too little correction.

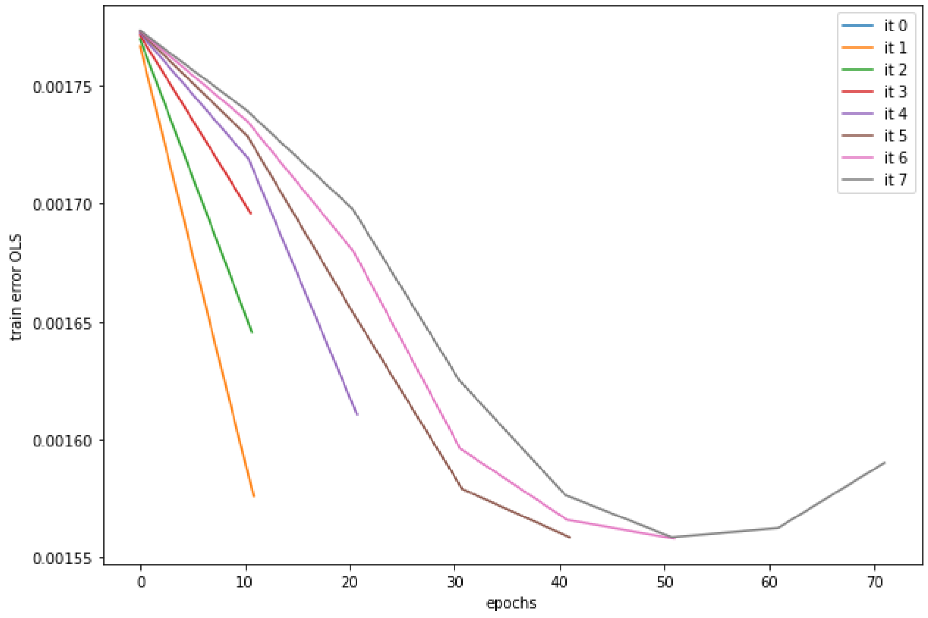

Figure 8: Error for 7 iterations of the iterative feedback training with the sigmoid model. Each iteration has more epochs and lower learning rate.

In Figure 8 and 9 we see the training error for seven iterations of the iterative feedback training. Here, each iteration has more epochs and a lower training rate. The reason for that is that we assume that each iteration will have better inputs, and we want to spend most of the computational resources on the best possible inputs. The weights are reset between each iteration. It is also assumed that the first iteration is quick and rough to get us out of the exact solution input space. However, both the first and second iteration appears to perform quite well. However, they start to diverge a lot faster and doesn't reach as low before diverging as the later iterations.

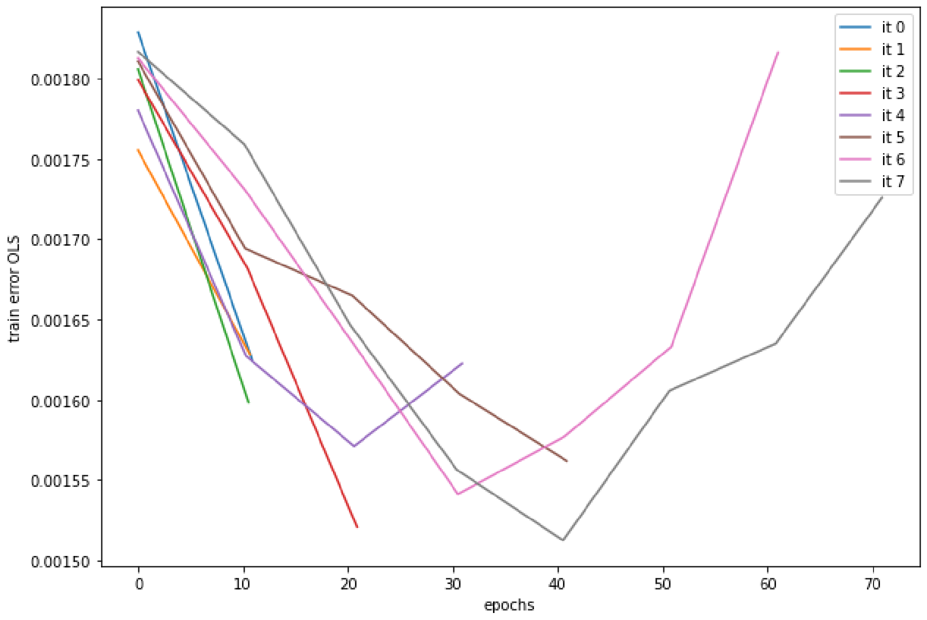

Figure 9: Error for 7 iterations of the iterative feedback training with the tanh model. Each iteration has more epochs and lower learning rate.

The reason for the divergent behaviour is unclear. It probably is due to the inherent limitations of the model, particularly the backpropagation. Lowering the learning rate seems to tame it, but only to a point as seen in Figure 8 and 9.

It is also unclear when learning is done. The approach used here is to try to stop the learning when the error is at its lowest before it starts diverging. However, this approach a major point of improvement.

3.6 Optimization and schedulers

As seen in Table 4 the model needs to compute the timesteps from $3.0E4$ to $3.0E6$ times depending on the lookahead. Because of this, it's exrtemely import for the timestepping to be fast, and for the model to learn as fast as possible. To acvieve this 2 things are implemented:

1. All the funcitons relating to timestepping (The whole solver and central difference scheme) have been rewritten in Numba [2] and compilen in nopython mode with parrallellization. Essentially this means that our solver is compiled to C, and used all threads available when training. The speedup was between a factor of 1000 to 100000 depending on how well the specific conditions was able to utilize all the computing resources available.

2. A momentum scheduler was used where the weights are adjusted according to

$$\Delta W = C_{mom} * \Delta W_{old} + \lambda W' \tag{26}$$

   Where ΔW is the change in weights, C_{mom} is the momentum constant which is the fraction of the precious change carried on to the next change, λ is the learning rate and W are the gradients of the weights.

[2] https://numba.pydata.org/

4 Results

Here, the network has been trained and tested on the sod shock problem. The configurations of the netwoek used are as described in Tables 1 and 2 who's only difference is the hidden layer activation function (sigmoid vs tanh). The initial conditions are as seen in Table 6

Table 6: Initial conditions for the sod test tube problem. Nx and Nt are the number of points in space and number of timesteps. x is the spacial axis evenly spaced and dt is the timestep. γ is the heat capacity ratio, ρ is the density and P is the gas pressure where L and R denotes the left and right side of the test tube.

Time and space				
Nx	$x \in$	Nt	dt	
256	$[0, 1]$	1500	0.0002	
Physical parameters				
γ	P_L	P_R	ρ_L	ρ_R
5/3	$1/\gamma$	$1.25/\gamma$	1	1.25

In Figure 10 the initial conditions are plotted for the parameters given to the network. As there is a large pressure and density discontinuity, a shock wave will propagate to the right. Over time the conditions stabilize and become uniform. We are interested in the first stage of this process, which involves the discontinuity and the shockwave, as these are phenomena that are notoriously difficult to model and a great test for computational solutions to hydro-dynamic systems.

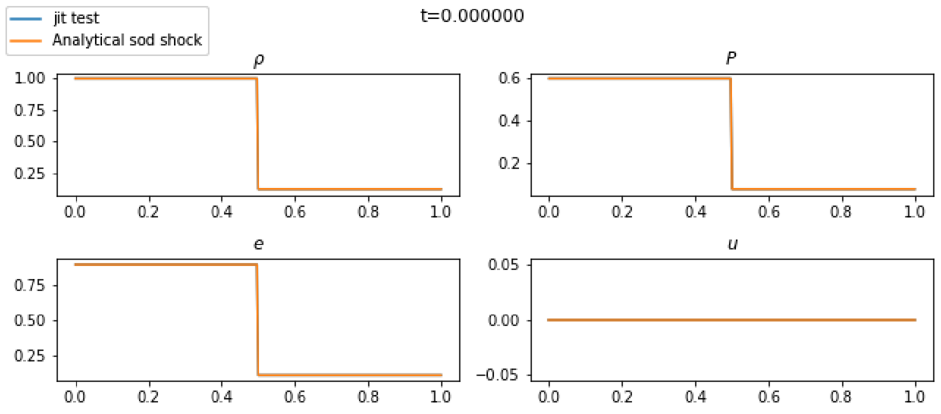

Figure 10: The initial conditions of the sod shock problem used in this project.

4.1 Comparing our model to other methods

First of all, we want to assess how much better or worse our models are than the other existing models in the field. Keep in mind that our model is based on a very basic central difference scheme, and should not be very good. Our neural network can only adjust the noise in the central difference with it's diffusice constant determination, and the network can't inherently change the method.

As a result we would want our model to perform better than setting the diffusive constants to be constant in time, but it shouldn't do better than more advanced methods such as Lax-Wendroff or Riemann solvers (Roe). The interesting part is how close can we get to the LW and Roe solvers, and is the tradeof of using a model like this worth it?

In Table 7 we see an overview of the best performing training iterations of our models versus the constant diffusion and the LW and Roe solvers. Our model performs significantly better than the constant diffusion both for sigmoid and tanh.

We get a closer look on which sections in time our model performs best in Figure 11. It is to be expected that the error accumulates and increases in time for our central difference scheme, as it does

Table 7: The mean OLS error over all 1500 timesteps for Roe, Lax-Wendroff, no diffusive constant, constant diffusive constants and our models predicted diffusive constants with central difference. For the predicted results these are the lowest values obtained for a range of different training parameters.

Name	mean OLS	Lookahead
$c_q = 0$ $c_L = 0$ $c_D = 0$	5.67E-3	
$c_q = 1$ $c_L = 1$ $c_D = 1$	1.78E-3	
$c_q = 2$ $c_L = 2$ $c_D = 2$	1.64E-3	
$c_q = 3$ $c_L = 3$ $c_D = 3$	1.93E-3	
Predict sigmoid	1.56E-3	1
Predict sigmoid	1.52E-3	50
Predict tanh	1.49E-3	1
Predict tanh	1.55E-3	50
Lax-Wendroff	1.19E-3	
Roe	1.12E-3	

not conserve quantities. Roe and LW however conserves the physical quantities in time, so it will not suffer from always getting worse. As a result, the LW actually performs quite bad in the initial stage, as it struggles to keep the shock front without oscillations.

While some of the constant diffusive models perform better either in the first or last stage, our model has the benefit of dynamically changing the diffusive constants to adapt in time, so we see overall better performance for both our models compared to any of the constant diffusion solvers in Figure 11. This was the goal of the network to begin with, so achieving this despite all the schenanigans with the unconventional backpropagation is considered a success.

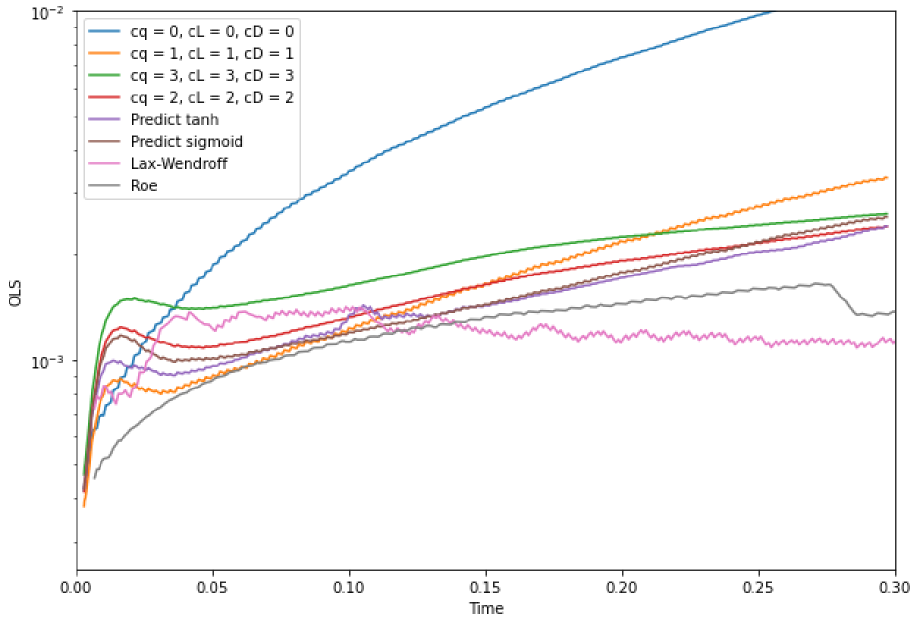

Figure 11: OLS error for 1500 timesteps for Roe, Lax-Wendroff, no diffusive constant, constant diffusive constants and our models predicted diffusive constants with central difference

4.2 The diffusive constants' change in time

Now that we have looked at the overall performance of the network, we will have a closer look at what it actually does to the diffusive constants. There is quite a difference between how our two models deal with the diffusive constants. Ideally, the neural network does as little as possible, and only does some minor adjustments where they are needed in the right place at the right time. Afterall, we are modelling a physical thing, and can't just make crazy corrections that imply unphysical behaviour.

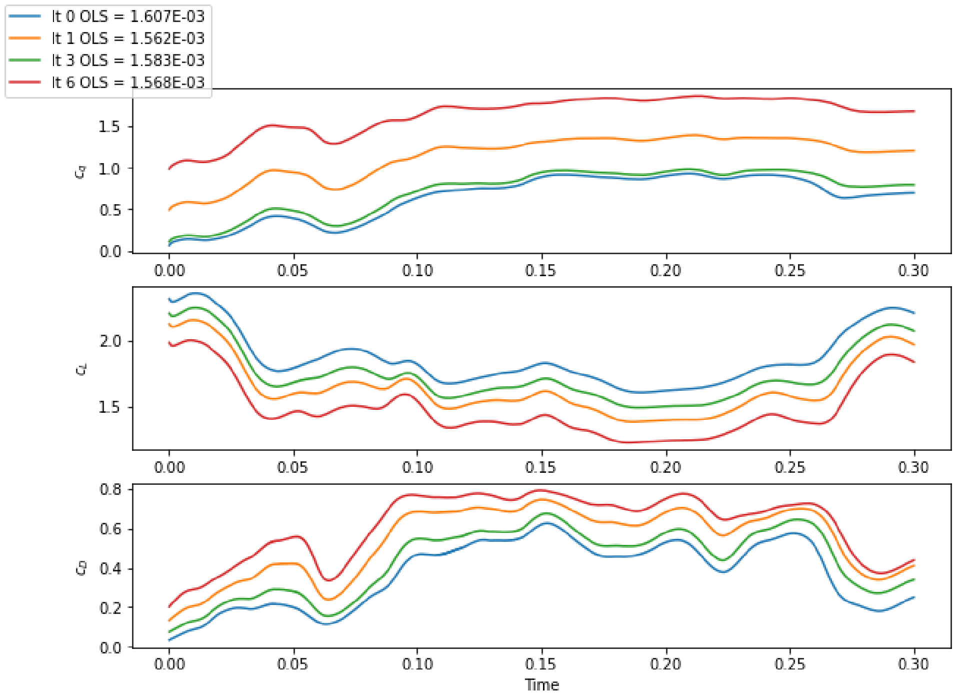

Figure 12: The diffusive constants determined by the sigmoid model for different iterations with no lookahead.

In Figure 12 we can see the change in time for the diffusive constants for different feedback iterations of the sigmoid model. Here, the changes are small, smooth, and the values are not extreme. We can also see that the different iterations generally have the same shape, as if it tries to preserve some ratio between the different constants. This ratio seems to be more or less the same, and gives some interesting insight into how the constants are interconnected in the timestepping process.

However, in Figure 13 the tanh model tells a completely different story with more rapidly changing and higher values of the constants. There does not seem to be any pattern between the iterations, and it is very hard to get any insight from it. The weird thing is that in the end the tanh model performs better than the sigmoid model. However, one might ask which one is more useful? Which one gives more insight?

To further explore the differences, the best performing iteration of each of the sigmoid and tanh models are plotted in Figure 14. It is clear that the models have learned very different things about how to set the constants, and there does not seem to be any correlation between them. Again this raises some questions. What type of learning are we interested in for machine learning in physics? Do we want to gain insight about relationships that are difficult to find, or do we want to brute force the smallest error regardless of whether the learning is insightful or useful?

Lastly, a snapshot of the predicted solution for the two models are plotted in Figure 15. Note that tanh has the smallest error. However, one could argue that the sigmoid better preserves the shockfront and overall shape of the analytical solution.

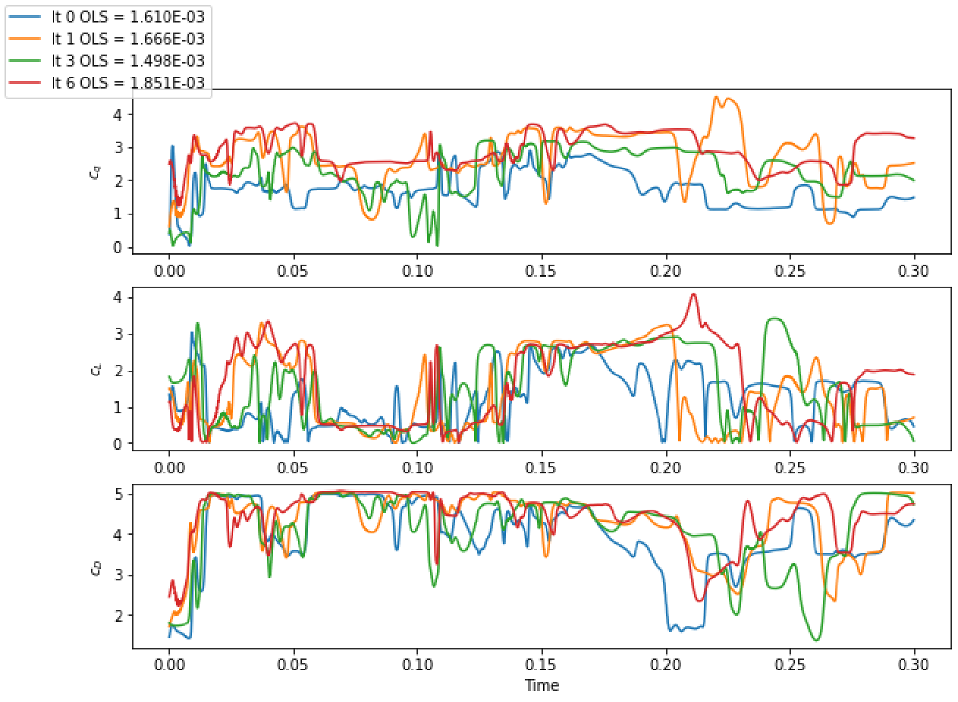

Figure 13: The diffusive constants determined by the tanh model for different iterations with no lookahead.

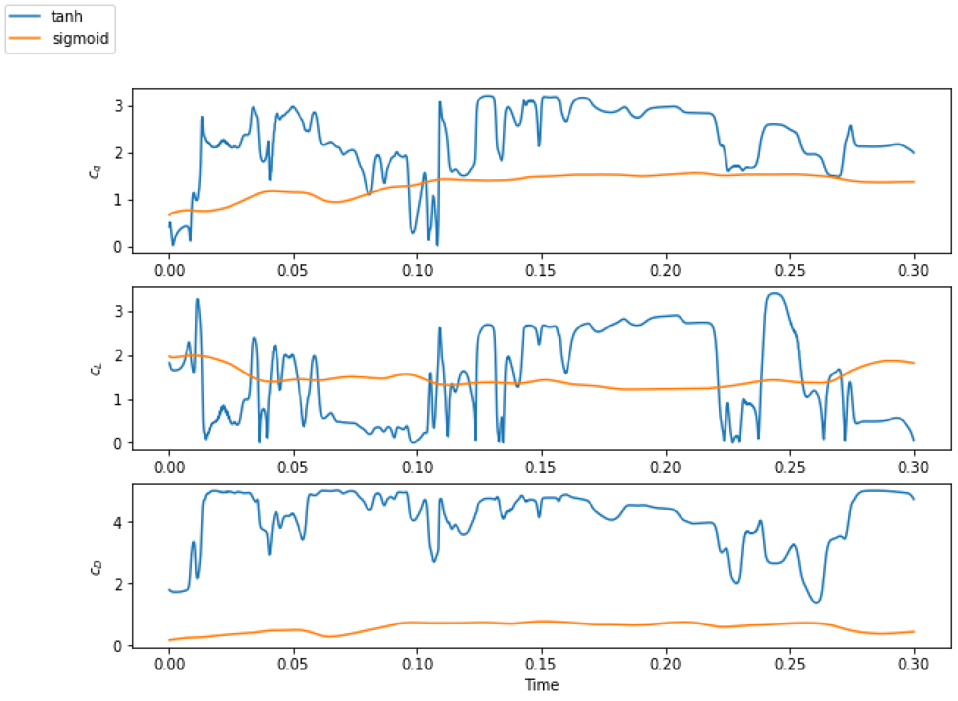

Figure 14: The diffusive constants determined by the best performing models of sigmoid and tanh with no lookahead.

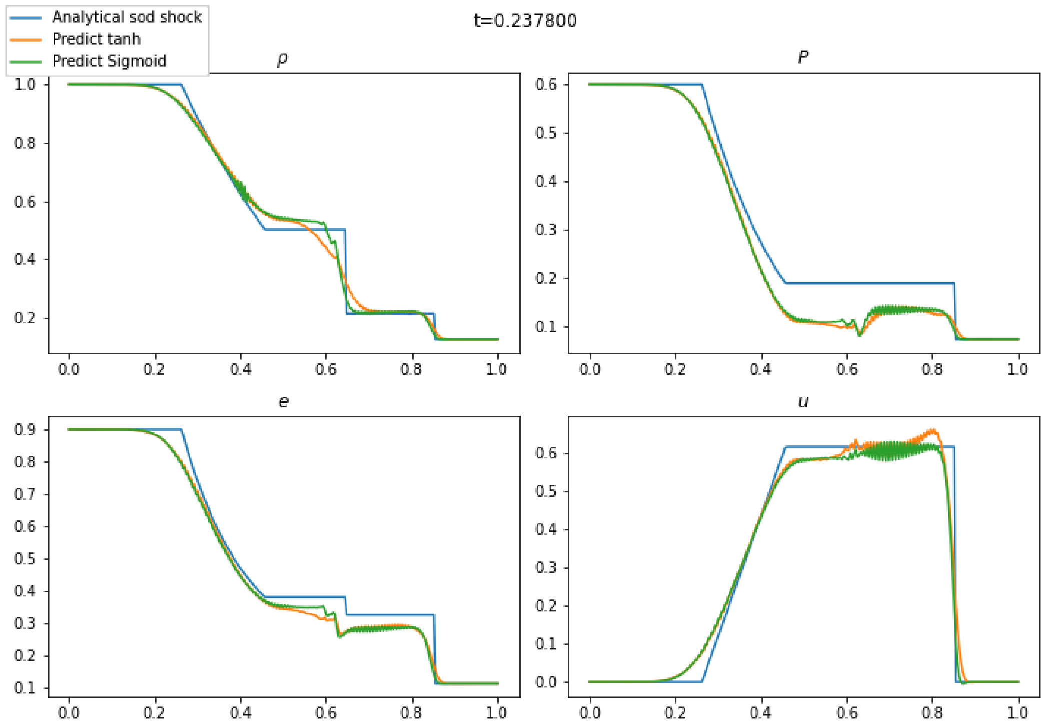

Figure 15: Comparing the best performing sigmoid and tanh models at the same point in time.

5 Discussion

5.1 Overall performance of the model, training and activation functions

Overall the model achieves the goal of setting the diffusive constants in a way that reduces the noise and improves how the shock-wave is handled. Additionally, training the network on its own output iteratively successfully helped the network learn how to handle builtup noise over several timesteps. At some point, iterating further becomes redundant and ultimately makes the model worse.

For the activation functions we see smaller overall error for tanh than with sigmoid. However, the tanh model does a lot more drastic changes and has higher values for the constants. Ultimately, we want the diffusive constants to be as low as possible, such that we don't actually change too much of the physics.

Another aspect that's subjectibely preffered about the sigmoid model, is that it seems to preserve some relationship between all the constants. Additionally, using it as aid to setting the constants at one specific value is a lot easier as the constants seem to lie within a smaller range. Maybe we can learn something about the non-linear way one diffusive constant impacts another? Perhaps similar applications can be used to understand other non-linear behaviour in physics?

On the other hand however, perhaps this erratic behaviour produced by the tanh model is just as valid as the sigmoid's smoother and more subtle approach. Perhaps the truncation error and numerical noise oscillations are not smooth, and need to be dealt with this way?

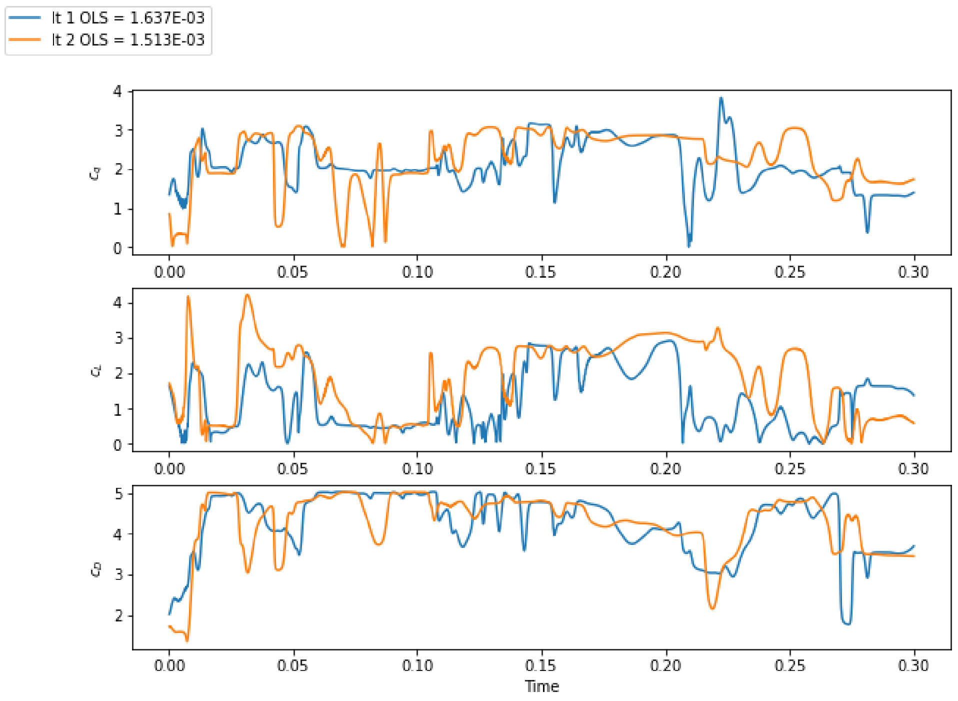

Figure 16: Caption

One of the reasons for implementing the lookahead was to reduce the rapid changes in the diffusive constants, however this did not work. In Figure 16 we see that the tanh model is just as crazy with 100 timesteps lookahead. Overall, the lookahed did help the models perform better, however it becomes very computationally expensive. Each epoch's training time scales almost linearly with the number of lookahead timesteps. Although the larger values of lookaehad did provide better results for the same amount of epochs, the number of timesteps performed increases the training time such that it is not really comparable. Overall the lookahead is not worth it in this scenario.

5.2 Optimization and runtime comparisons

One of the aims of this investigation is to use neural networks to better utilize computing resources for plasma simulations. Naturally, we would want to compare the runtime of using a central difference scheme with and without the neural network and the flux conservative methods. However, all of these methods are implemented with varying degrees of optimization across standard Python, numpy and numba. Comparing them would at best give a measure of which one is better optimized. It is outside the scope of this investigation to make a detailed and fair same optimized comparison. However, looking at the complexity of the operations in the different algorithms, we can make some educated guesses.

- The central difference solver is fastest being the most primitive.
- The central difference solver with the neural network is slower, it all depends on how fast the feed forward algorithm is and how many hidden layers there are.
- The LW algorithm is slightly more complex than the central difference scheme, but has no matrix inversions, dot products or other tedious linear algebra operations.
- The Roe scheme needs to do a matrix inversion and 3 dot products with 3 matrices of dimension 3x3 for every spacial point every timestep. This is notably slower, and needs several seconds to finish while the other algorithms finish almost instantly.

Again we have to stress that none of these algorithms are optimized except for the central difference scheme as it is used in the training of the network. That being said, we have managed to take an extremely primitive numerical scheme and improved it significantly with the assistance of the neural network and some diffusion. Imagine how this could work for a more accurate finite difference scheme and a more mature implementation of neural network assistance. Perhaps that's the future of numerical simulations?

5.3 Design issues and areas of improvement

There are several issues with the design used in this project, which is a result of the explorative nature of the investigation. Here are the most pressing ones listed:

1. The derivative of the cost function has the wrong dimensions with a whole numerical solver whose derivative is unaccounted for. This leads to dubious calculations for the weights.
2. The way the (ill calculated?) gradient of the cost function is reshaped from 1024 to 3 points is a massive over-simplification of the non-linear effect the diffusive constants have on each other and the solver-network system.
3. The cost function only cares about how closely the solver approaches the values of the analytical solution in each point on average. It does not care about keeping the shocks intact or the overall shape of curves.
4. If the network decided to do very large adjustments to compensate for the shortcomings of the numerical scheme it won't be penalized. This is an issue because we want the network to do as little as possible; just enough to reduce noise and artifacts from the shock and discontinuity.

The first two issues listed over are the major areas of improvements. Finding a better way to tell the network in what way the weights should be adjusted is the Achilles's heel of this project.

Another huge shortcoming is that the training error converges for a while, reaches a minima and then diverges. This is almost certainly a symptom of the already mentioned issues with the gradients, and it might be something very small, or something inherently problematic with the approach used. While working on the model, a lot of different things were tried to fix this divergent training error. However, they either made the model worse or made it diverge right away. It is concluded that this issue is a symptom of the other issues, and not something that should be treated directly.

5.4 Suggestions for further work of on the model

In addition to making improvements on the already mentioned issues with the model, there are also new features that can be added to the model which may prove to be useful. In this project, the main way we tried to introduce time was through the lookahead. However, we might be better of by implementing a recurrent neural network. With recurrent neural networks we allow the network to remember what happened several cycles before. In our case, this might mean that the network can discern whether it is just moving along a shockwave that's supposed to run it's course as unaffected as possible, or whether it some extreme peak or divergent unstable behaviour is evolving. These two different scenarious require different treatment, where one should be preserved and one should be dampened/diffused. As of now, the network does not know the difference as it only has access to the current state of the system.

Further, one or more convolution layers to try to filter and extract the most important parts of our inputs could help our network learn faster and be more focused. As we are interested in diffusing unstable oscillations these could be detected by doing a derivative in a convolution layer. Also, we could highpass for small oscillations due to numerical noise assuming they happen on smaller wavelengths.

Another area where convolution could be implemented is in the reduction of dimensions for the gradients of the cost function. The solution in place now with Table 3 is a massive oversimplification, and as it is non-trivial to know the best way to do this. In image processing filters and maxpooling are commonly used to reduce dimensionality [11]. Perhaps setting up a second network for this process could be useful?

Before starting the imlpementation of the neural network in this project we exlpored the use of Fourier Transform to look for oscillations in time and in the spacial axis. Both were effective in identifying the type of numerical noise and oscillations our algorithm is trying to reduce. This further solidifies the potential of convolution layers in this type of problem.

5.5 What can be expected from a model like this?

Now that we have discussed the major areas of improvements on the existing design, as well as possible additions, we want to assess what we should be able to expect from our model. Ultimately, the model is help hostage by how bad the central difference scheme is at the sod shock tube problem. Our neural network can only diffuse numerical instabilities and it can't fundamentally change the way the numerical scheme works (or doesn't work). This was done on purpose, because if we can make a really simple and useless numerical scheme useable, then perhaps we can make a good numerical scheme great with a similar approach elsewhere.

Consider this project a proof of concept, where we extracted a difficult and non-linear relationship from a set of equations (how the diffusive constants impact the noise reduction in coupled equations) and taught a neural network how to only solve that part for us. Actually approximating the solution itself was done by the well known numerical scheme, and the neural network only makes adjustments. As a result, we should be happy that the solver with the neural network assistance performs better than no assistance.

Going back to universal approximation theory, we have a deliberate discontinuity in our data, both in the pressure and density. It is unclear how this discontinuity propagates through the network. Additionally, the discontinuity only appears in the initial condition of the problem, and for the remaining 1499 timesteps there are no discontinuities.

For some cases of universal approximation theory we also require the cost function to be differentiable. In our case we have the numerical scheme between the network output and the cost function, and although that might be differentiable it certainly is not differentiated in the current implementation.

It is simultaneously incredibly exciting and very unsatisfactory that the model to some extent works with such poor understanding of how it works. Specifically we are referring to the cost function, it's derivative and how the gradients are updated.

6 Conclusion

Our exploratory work has proven that the neural network and finite difference scheme combination can produce better results than only using the scheme as seen in Figure 11 and only a neural network[4]. Additionally, we have demonstrated with the two different activation functions that depending on the setup some insight can be gained about the underlying physics the model is learning. For the sigmoid model, we can both extract some optimal ranges for the diffusive constants and see how they all change together, seemingly maintaining some proportionality between themselves. However, this does not replace actual physics, and perhaps the way the sigmoid model looks is a result of the shortcomings of the model, and should not be used to infer any physics. Yet again, the fact that the tanh model does behaves differently suggests that it's not completely determined by the design of the model. Further, we have identified and thoroughly discussed the shortcomings and challenges of this type of model. The majority of the improvement is believed to come from improving the gradient of the cost function and finite different scheme layer. The divergent learning behaviour is also thought to be a symptom of the same shortcoming. With the predicted diffusive constants the solver performs better than no diffusive or constant diffusive constants. For the first half of the simulation, our model also performs better than the LW scheme, which is the more lightwight conservative scheme, and is only outperformed by the heavyweight Roe scheme. For the second half both the conservative schemes perform better, but overall the performance of our model is considered a success. Lastly, our model is held hostage by how bad the central difference scheme used is for the sod shock problem. Implementing a similar marriage between a better finite difference scheme and a more advanced neural network might prove very fruitful. Regardless this investigation considered a proof of concept for a type of model that might have a lot of potential.

7 Acknowledgements

The model is based on the feed forward neural network provided in the course-material github[3]. Additionally, the hydrodynamic codes are modified (mostly optimized for speed) implementations of code written for the course AST95510 Numerical Modelling at UiO [4]. Further, the analytical solutions are using a python library called sodshock[5]. The implementation of the Roe and Lax-Wendroff solvers are modified versions of P. S. Volpiani's implementations who is so kind and publishes youtube videos and code examples on fluid dynamics [6]. Without the above mentioned resources this project would not be possibe.

The codes for this project can be found in the repo https://github.com/GauteHolen/FYS-STK9429,

8 Comments on the assignments

Lastly, I'd like to share some comments on this project and its difficulties as you've been interested in this in previous projects in your courses. Firstly, a lot of time was spent on the overall design of the model and implementing the code and connecting everything together as seen in Figure 6. Initially, I was not aware of the shortcomings of the design, particularly those about the cost function derivatives. Narrowing down the shortcomings to one specific part of the code and trying to not only find a solution, but find out why finding a solution is very difficult, took a lot of time. This time was spent researching solutions, trying something new, being confused as to why it doesn't work, breaking the problem down to it's most basic parts, being confused again and then trying something else until eventually something worked. I would have loved to spend a lot more time before writing this report. Handing in something with some good results from the model but also a dubious method for implementing the solver with the cost function and gradient updates feels bittersweet. On one hand it's cool to do something new and explore, but on the other hand I would have liked to achieve a more rigorous understanding of how to fix what's wrong with the implementation. I guess it's a good proof of concept, and it has helped me learn a lot, which ultimately is the point of the course. But, I would like to actually figure out how to make this work a lot better, and also understand why it works. I think these types of hybrid neural network and finite difference schemes have a lot of potential.

References

[1] F. Harlow, "A machine calculation method for hydrodynamic problems," *Los Alamos Scientific Laboratory report LAMS-*, 1956.

[2] G. V. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.

[3] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[4] C. Chang, L. Liu, and T. Zeng, "Finite difference nets: A deep recurrent framework for solving evolution pdes," 04 2021.

[5] A. Kratsios and I. Bilokopytov, "Non-euclidean universal approximation," *CoRR*, vol. abs/2006.02341, 2020.

[6] G. A. Sod, "Review. A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws," *Journal of Computational Physics*, vol. 27, pp. 1–31, Apr. 1978.

[7] S. T. Miller, J. F. Lindner, A. Choudhary, S. Sinha, and W. L. Ditto, "The scaling of physics-informed machine learning with data and dimensions," *Chaos, Solitons Fractals: X*, vol. 5, p. 100046, 2020.

[3] FYS-STK 9429 FFNN code
[4] nm_lib github link
[5] sodshock python package
[6] P. S. Volpiani github with codes

[8] U. frisch, “Translation of leonhard euler’s: General principles of the motion of fluids,” 2008.

[9] J. VonNeumann and R. D. Richtmyer, “A Method for the Numerical Calculation of Hydrodynamic Shocks,” *Journal of Applied Physics*, vol. 21, pp. 232–237, 04 2004.

[10] L. Margolin and Lloyd-Ronning, “Artificial viscosity -then and now,” 02 2022.

[11] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson, *Handwritten Digit Recognition with a Back-Propagation Network*, p. 396–404. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990.