Analysis of Physics informed Neural Networks applied to the 2D Acoustic Wave Equation in Complex Media

Sjoerd Akkermans

Bachelor Technische Natuurkunde TU Delft

Delft, 17 November 2021

Supervisor: D.J. Verschuur

Contents

Abstract

In this thesis we have analysed the behaviour of a physics informed neural network and it's competence in predicting a wave in a non-homogeneous medium. During this project we have used a fully connected network with labelled input data of a 2D acoustic wave. On top of this we used a special loss function that calculated whether the output of the network satisfies the wave equation. Our experiment consisted of the tuning of the hyper parameters, analysing the optimal choice of activation function and the optimisation of the input data and improving the loss function. During this project the unpredictable nature of machine learning has become very clear. We have experimented with several activation functions and have found that the optimal choice of activation function depends on how long you are willing to train the network, as the development of the loss function differs immensely between activation functions. When we looked at the optimal scaling of the input values we find that a non-trivial scaling seems to work better than for example, normalisation of these values. Furthermore we have tried to improve the sampling of the points we use to calculate whether the prediction of the neural network satisfies the wave equation and got interesting results. When we implement all optimisation techniques, we find that the neural network is extremely capable of predicting the wave's behaviour in a high contrast media within the time frames of the input data. Prediction outside of this time frame does work but the results do deteriorate especially in the positive time direction. Predicting in the negative time direction yield slightly better results.

1. Introduction

The wave equation can be extremely hard to solve in complex media. In these kinds of media the wave gets reflected and transmitted a lot and because of the differing wave velocities, a wide variety of wavelengths and wave fronts exists. It is possible to simulate this behaviour using a finite difference model but this is a timely task and has it's limitations[1]. Machine learning could offer a much more efficient approach than the current techniques. Lately there has been a lot of public interest in machine learning, this is partly because the field benefits from a lot of exotic sounding names, ranging from "neural networks" to "deep learning" terms that seem to grab the imagination of many people as well as Hollywood. There are also some valid reasons for this enthusiasm. Thanks to the development of computers and wide spread access to state of the art machine learning libraries, machine learning has become a very accessible field to whoever is interested.

During this thesis we will utilise these developments, we will create a Physics Informed Neural Network, or PINN for short. This is a type of supervised learning that uses not only the classical way of determining the correct prediction of the network, by comparing it to the input data. But also compares the predicted output of the network to the underlying physical principles, in our case the wave equation. Currently most of these simulations are done using finite difference simulation, this type of simulation is widely used to solve differential equations. A down side to this method is that the calculation has to be done on a grid and discretised. Another problem with this technique is that in order to know the state of the wave at a certain point in time you would need to calculate all time-stamps before that.

In this thesis we will try to analyse and attempt to improve a physics informed neural network, the network previously worked on by Jesse Buijs [2]. We will do this by investigating what activation function work best and how combining these affect the results, data processing of the input data, and improving the method the physics loss is calculated.

In Chapter 2, we discuss the underlying theory. Chapter 3 explains our the experimental method. Chapter 4 shows the results of the experiment and finally in chapter 5, we draw conclusions and give recommendations for possible further research.

2. Theory

2.1. Neural Network

A neural network is a programming technique that is inspired by the networks found in the brains of animals and humans[3]. Essentially it is an abstract way of statistical pattern recognition that allows computers to "train" on a given data set and find the underlying patterns. In this case we are using a supervised network. A supervised neural network is given a labelled input data set[4]. A common example would be a dog/cat classifier. To train this neural network you would provide a set of images, the input data, and a label for each image whether it is a dog or a cat. After letting the neural network train on this training data, you could input a new image, not yet seen by the neural network, and the neural network could predict whether a cat, or a dog would be in the picture. A deep learning neural network, a neural network that consists of 3 or more layers. These layers between the input and output layers are known as "hidden" layers. Each of these layers consist of a set of "neurons". The basic structure of a neural network is shown in figure 1.

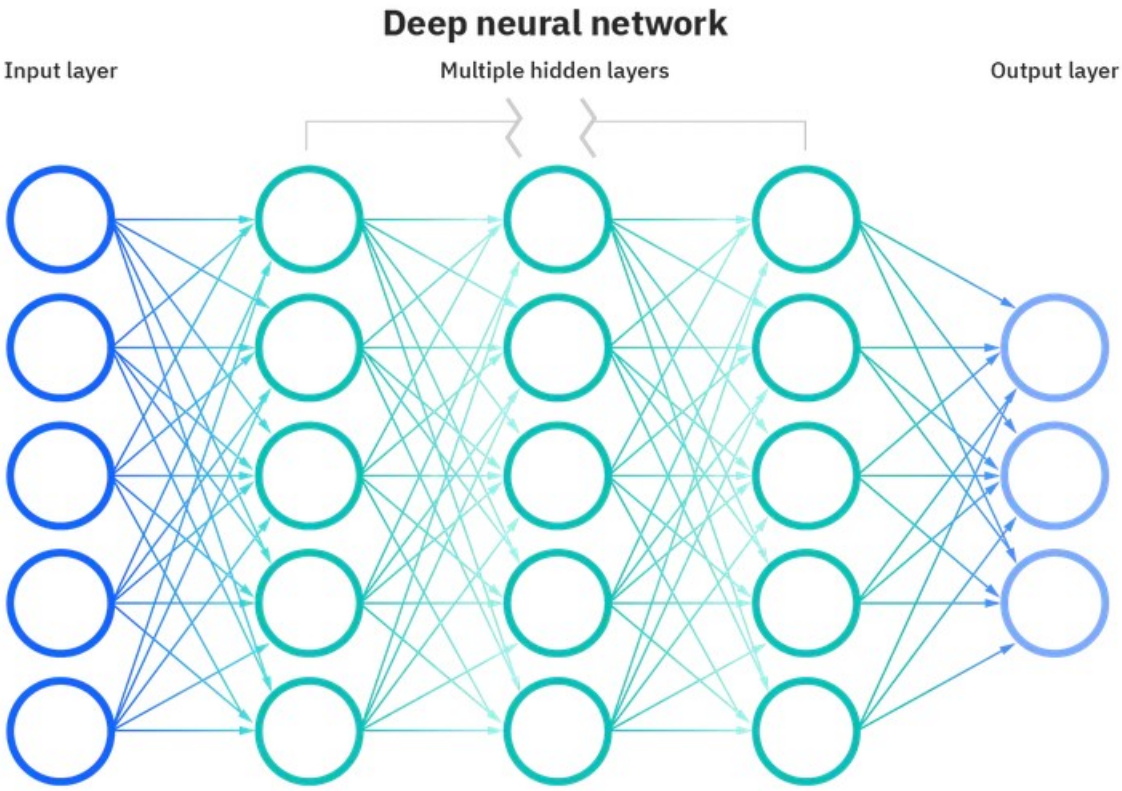

Figure 1: Basic structure of a fully connected deep neural network.[5]

2.2. A Neuron

A neuron takes in a set of variables, in our case this is equivalent to the amount of neurons in the previous layer, and linearly scales each of these values with a certain weight and adds a bias.

This resulting value is then passed through an activation function. An activation function is, except in some special cases, a nonlinear function that converts the input values in a non-linear output values [6]. Activation function can take a wide variety of shapes, but often it is desirable if it's a differentiable function. A few examples of activation functions are given in 3.4. The output of the neuron k of layer l can be expressed as follows

$$X_l[k] = f(W_{l_k} X_{l-1} + b_{l_k}) \tag{1}$$

where W_k is a $n \times 1$ matrix containing the weights and X_{l-1} a $1 \times n$ matrix containing the output variables from the previous layer and b the bias, a scalar. $f(x)$ is the activation function which varies.

2.3. The Loss Function

After the last layer of the neural network we calculate the error in the output of the network, we call this the loss. This can be defined in different ways and enables you to guide the learning process. An example of one of the more common and simple loss functions is the so called L1 function[6]:

$$J_{L1} = |Y^{'} - Y|, \tag{2}$$

where Y' is the output of the last layer and Y the desired output, (the label of the labeled data). We will be using a combined loss function consisting of a L2 part and a Physics-loss part. The L2 loss function is given below with the same parameters as the L1 loss function.

$$J_{L2} = |Y^{'} - Y|^2. \tag{3}$$

The physics loss is where the physics informed part of the PINN comes in. This physics loss part not only ensures that the neural network produces a wave that is optically close to the desired wave, It also forces the output to satisfy our physical constraint, the wave equation:

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial z^2} = \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2}. \tag{4}$$

We use this function as the basis for the physics loss function. Rewriting it a bit and taking the absolute value gives us:

$$f_{ph}(p) = |\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial z^2} - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2}|. \tag{5}$$

Since we are not concerned about just the loss of one input data set, but more on how well the algorithm predicts the pressure for all data sets we take the average of all the loss values of all input values, resulting in the following equation for the total physics loss:

$$J_{phi} = \frac{1}{M_{ph}} \sum_{i=1}^{M_{ph}} f_{ph}(p^{(i)}). \tag{6}$$

Were M_{ph} the amount of points where the physics loss is calculated and J_{phi} the total physics loss.

2.4. Backpropagation

During training this loss is then used for fine tuning of the weights for each neuron, with a process known as back propagation. This technique determines the "gradient" of each neuron on a per layer basis. First we determine how the loss changes with the output of each neuron, then we determine how much the neuron output was influenced by the bias (b) and the weights (w). Using the chain rule we can easily repeat this process on a layer by layer basis [7]. Concretely for a 1 neuron network it works as follows: We start at the output layer and plug in the values for W and b, at the start these are set as small random values. Then using a process know as "Forward propagation", or just plugging in the numbers, we determine the Loss for these values. The next step is to determine the influence of the output on the loss, this is done by calculating the derivative of the loss function with respect to the output value. For the L_2 loss function this is shown in equation 7.

$$\frac{\partial loss}{\partial Y^l} = 2(Y^l - Y) \tag{7}$$

Where Y is the correct prediction, Y^l the output of the neuron at layer l.
Next up is determining how much each parameter (W and b) influenced this output, this can be calculated with the chain rule as shown in equation 8.

$$\delta_{layerl} = \frac{\partial Y^l}{\partial w}\frac{\partial loss}{\partial Y^l} \tag{8}$$

With δ being the gradient. In this case it is done for a weight but for the bias it is analogous. Since all neurons within a layer are independent from each other we can calculate this for every weight of every neuron for that layer. Once we have determined the gradients for all neurons in a layer we propagate one layer back and do the exact same for that layer. This is easily done because once again we can use the chain rule. As shown in equation 9.

$$\delta_{layerl-1} = \frac{\partial Y^{l-1}}{\partial w}\frac{\partial Y^l}{\partial w}\frac{\partial loss}{\partial Y^l} \tag{9}$$

Once all gradients for all weights and biases for all neurons have been calculated you can update these values, all at the same time. The stepsize with which you update the values is know as the "learning rate". If this is too big, the network might overshoot the optimal weight value, if it is too small it takes a enormous amount of time to train the network. Mathematically this can be expressed like in equations 10 and 11.

$$w_{new} = w_{old} - k\delta_l \tag{10}$$

$$b_{new} = b_{old} - k\delta_l \tag{11}$$

Where k is the learning rate, w the old and new weights and δ the gradients.

3. Experimental Method

3.1. Training data

For this experiment we used computer generated data. First we created a so called velocity profile, this is essentially a two dimensional map that defines the wave velocity on every point, an example of such a map is shown in figure 2.

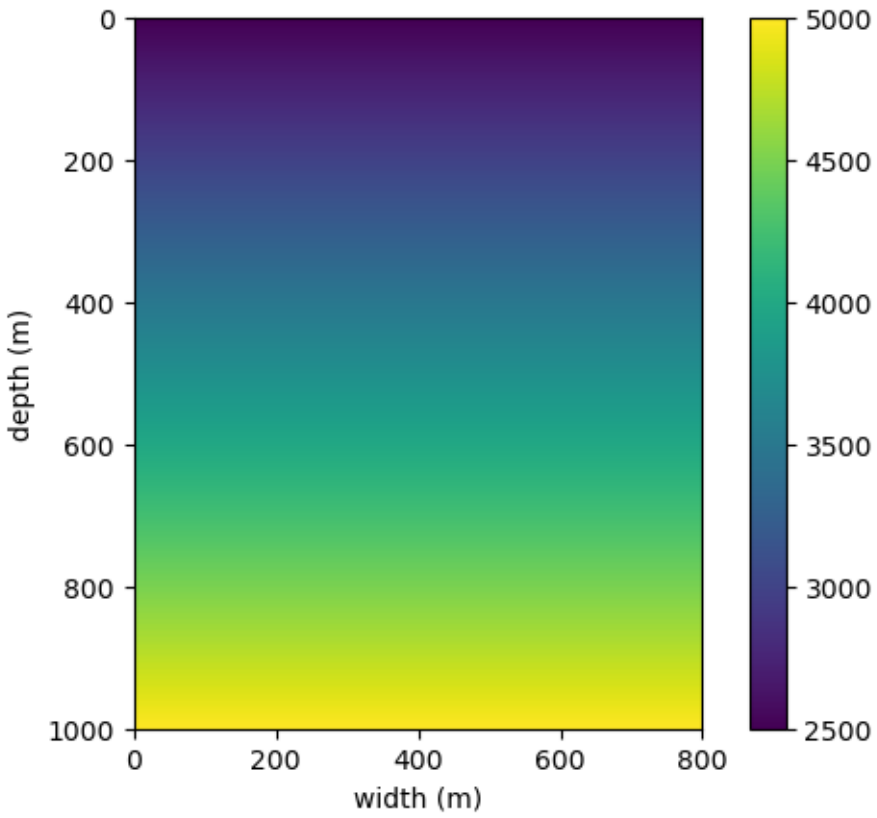

Figure 2: Example of a 2D velocity map, the unit of the velocity is m/s.

After having created this velocity map we simulated the behaviour of the wave using a finite difference model. The results of this simulation are pressure values for 5 dimensional coordinates: x coordinate in meters, z coordinate in meters, time (t) in seconds, wave velocity (v) in meter per seconds, and density (ρ) in kilogram per cubic meter. This simulation data can then be used for the training of our neural network. This data is not directly inputted into the network, it will be sliced, a process where we remove certain parts of the data, and scaled. This gives us a few degrees of freedom to change the results and training time of the network.

Slicing is a technique where not all training data is offered to the network and parts that slow down the training or even degrade the final results are cut off. This can be done in all 5 input dimensions (x,z,t,v,ρ) and we have done it in 3 dimensions (x,z,t). During the entire experiment the data has been sliced in the time dimension to be limited to 50 different data sets spread evenly over a range of 100 time stamps. This has been done for two reasons. The first reason is to cut off edge cases, timeframes where the simulation is inaccurate because of the limitations simulation, examples can be seen in figures 3 and 4.

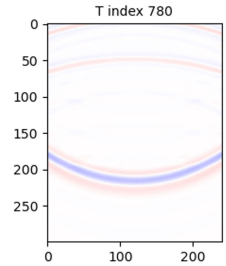

Figure 3: Edge case with little to no valuable information for the network.

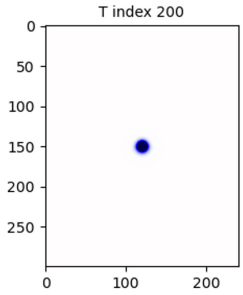

Figure 4: Edge case with little to no valuable information for the network.

Secondly, we increase the time step between input data and therefore cut off parts of the data. We do this because it's very similar to the other data and provides little improvement to the overall results of the neural network. A similar reasoning is used for the slicing in the x and z dimensions. Because we are only interested in the behaviour of the wave, the parts of the data where the wave is not (yet) present are simply not of interest of us, this null data only slows down the training process and even can reduce the quality of the results.

Scaling is a more nuanced technique of processing the raw data, it can be done in a different ways and the results are harder to predict. Part of the research done during this project has been done in optimising this scaling. We have experimented with scaling of the input variables and it's label (pressure P). We have scaled (P,t,v,ρ) independently and (x,z) with a shared scaling factor.

3.2. Configuration of the neural network

Like we mentioned before, we are using a supervised network, where each layer is fully connected to each other. The network consists of 6 hidden layers with 80 nodes each. Other constant parameters can be found in table **??**.

Table 1: Constants of the neural network.

Learning rate	2.5 e-4
Batch size	500
Test/Validation Ratio	1.86
Input Features	5
NN/PINN ratio	10

3.3. Defining our dynamic loss function

The loss function for this experiment is not static, the first few hundred epochs it runs with a simple loss function (L2) and after this first run the physics loss part will be enabled and will train about 10% more epochs. This is done because we don't just want the output of the network to satisfy the wave equation, but satisfy precisely the right solution of the wave equation that matches our finite difference data, that acts as "boundary condition" to the network. Since calculating the physics loss is a much more time consuming process involving much more steps then just calculating the L2 loss, it also saves us a lot of time by running most epochs without the physics loss part.

As can be see in equation 12, there is the factor P_{inc} that scales the losses relatively to each other, during this project we have experienced that this factor plays a big role in getting the optimised results.

$$J = J_{L2} + P_{inc} * J_{ph} \tag{12}$$

For example, if the physics loss would be much bigger and thus much more influential than the data loss, it could force the network to output the simplest solution to the wave equation, the trivial solution in this case, where the pressure is zero everywhere. Like we already mentioned, it takes a lot of time to calculate the physics loss. Therefore we only calculate the physics loss for a limited amount of points. Which points you choose to do this heavily influences the results of your network. During this experiment there have been two techniques used for the sampling of these points, the technique used by Jesse Buijs[2], Latin hyper cube sampling, and the technique we used which we used which goes as follows. We pick a random data point, when it's pressure value is above a pre-determined threshold it will be added to the list of points, when it's below the threshold it has a 20 % chance to be added to the list of points. This process continues until you have a the desired amount of data points, these points are then used in the physics loss function.

3.4. Activation functions

Another part of the project has been on analysing what activation function works best for our network. A problem that became clear directly from the start of this experiment is that the loss seems to be stagnant for some period and then suddenly starts to drop again, clearly visible in figure 5. This problem is often caused by the fact that during the process of back propagation the change in certain weights is so small that the entire networks seems to "wait" on the required change to happen. This problem is known as the vanishing gradient problem [8]. Figure 6 shows the development of the weights that have reach such a point of stagnation.

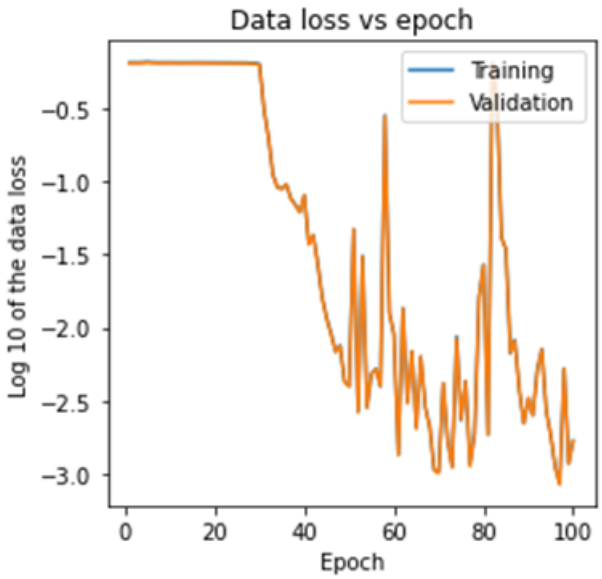

Figure 5: Stagnation of the loss with a sudden drop after that.

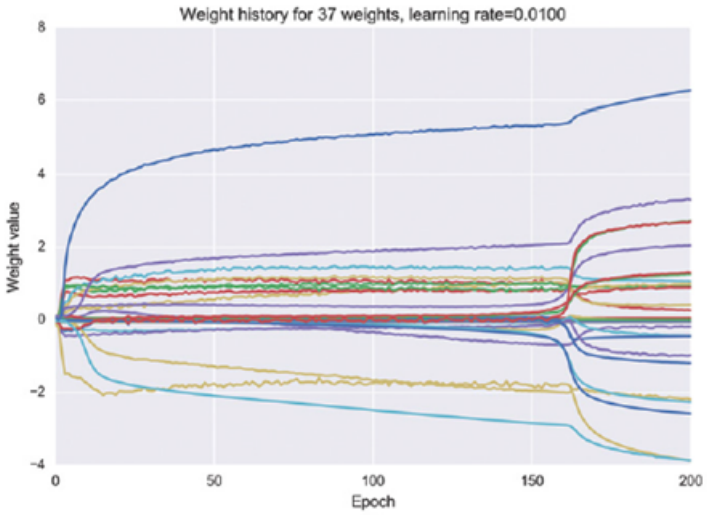

Figure 6: Literature example of weights during stagnation of the data loss[4].

What becomes really clear in figure 6 is that the even though the loss seems to be stagnant, the weights are not. They are just moving in such a slow pace that it takes a large number of epochs before making a dent in the data loss. The gradient of the activation function has a direct influence on this phenomenon and, therefore, we have experimented to find what kind activation functions gave the best performance and whether combining these would yield interesting results. We have looked at 4 activation functions, ReLu, Leaky ReLu, Sigmoid and Softplus.

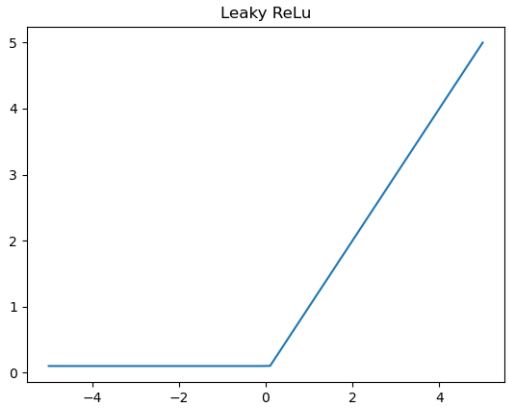

Figure 7: The Leaky ReLu function.

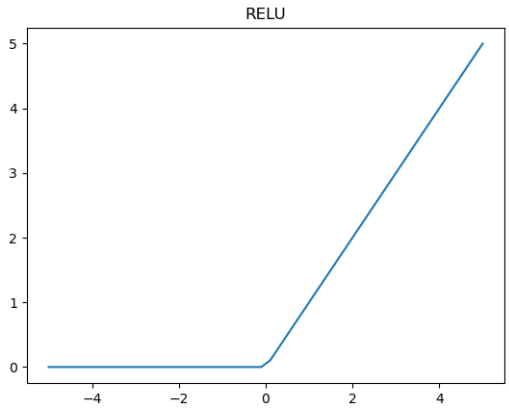

Figure 8: The ReLu Function.

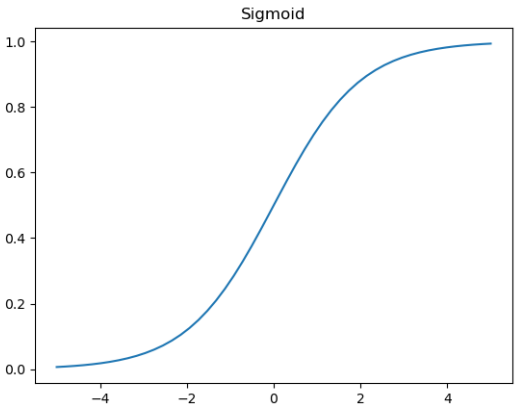

Figure 9: The Sigmoid Function.

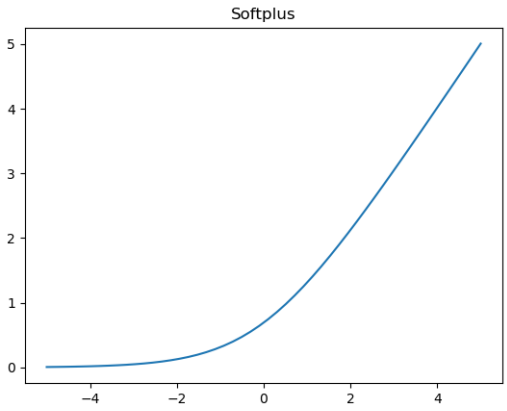

Figure 10: the Softplus function.

4. Results

In this chapter we discuss the results found through-out the project. The order we do this in is based on the the amount of influence each parameter had on the behaviour of the network. We will begin by discussing activation function, then we will continue with the scaling of the input variables. After that we will show our findings regarding the physics loss and finally we look at how the network performs with very long training intervals.

4.1. Activation function

We start by investigating what would be the optimal activation function. As can be seen in the figure 5 the initial network seems to suffer from vanishing gradients [8]. We find that when we use a ReLu activation function, this problem seemingly disappears, as shown in figure 11. Sadly this improvement is only temporary, as is shown figure 12 the initial gain diminishes after

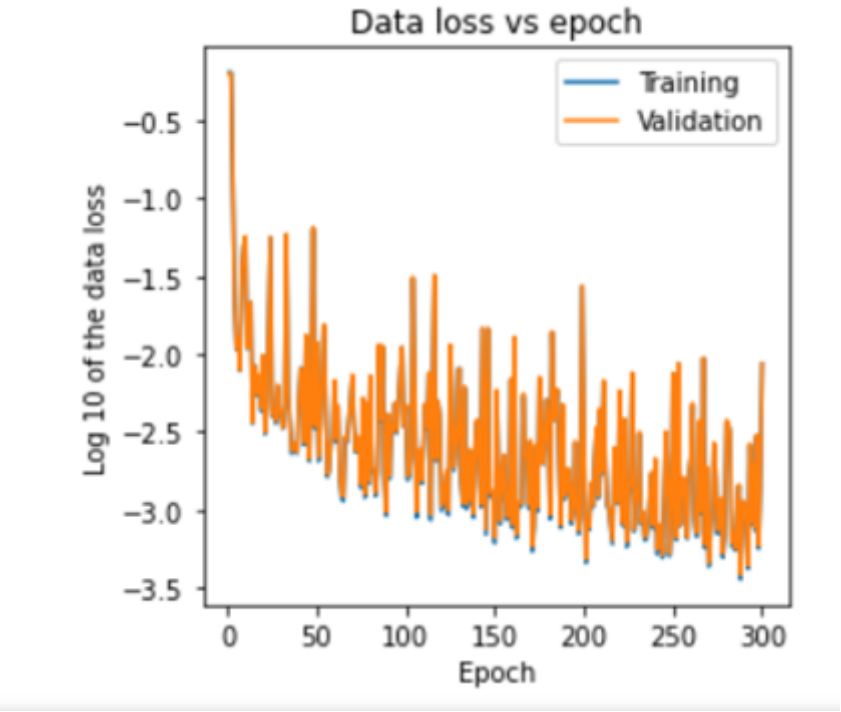

Figure 11: 300 epochs of the neural network with a ReLu activation function, PINN is disabled.

a few hundred epochs, therefore the soft plus activation is still superior to the ReLu function.

We have also experimented whether combining activation functions would yield better results. Even though this has little theoretical rationale, we did do experiment with it. As behaviour of neural networks can be very unpredictable. What we have tried is implementing a leaky ReLu that starts as a ReLu and then gradually increases it's "leaky" part. We tried switching the activation function with a similar one halfway through the training process. This all yielded significantly worse results. In figure 13 this behaviour can be observed.

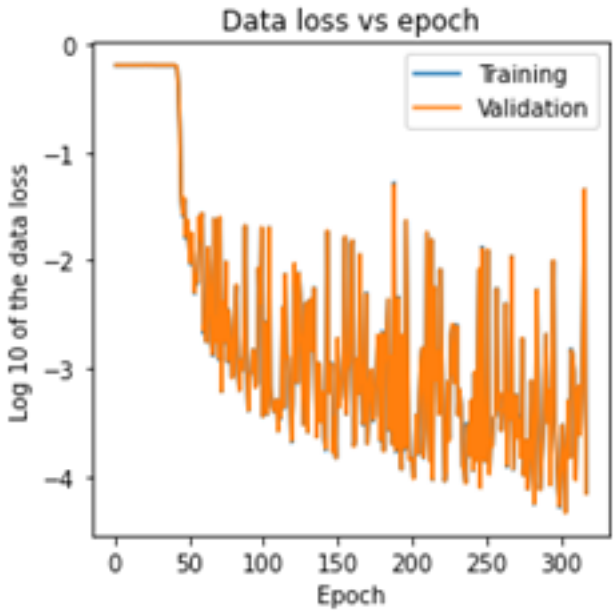

Figure 12: 300 epochs of the neural network with a softplus activation function, PINN is disabled.

Figure 13: An example of a the development of the loss function after 120 epochs. In this case the network started with a ReLu function.

4.2. Scaling

Another part of our project has been focused on investigating what scaling would be the optimal, for what range of values does the network create the best results? Common practice for machine learning is to normalise the input parameters, so we have tried this. The results of the first 100 epochs with normalised input can be found in figure 14.

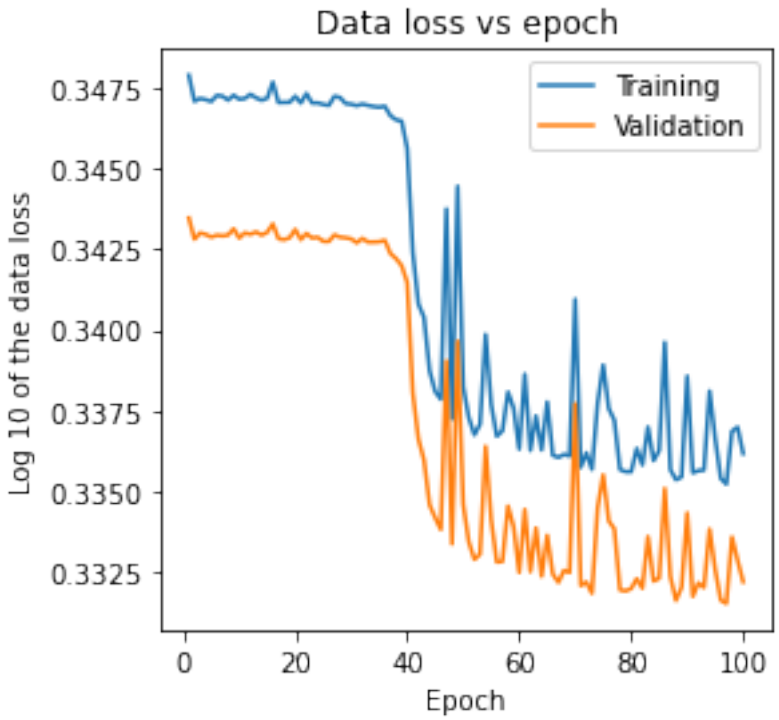

Figure 14: 100 epochs of the neural network with normalised input values.

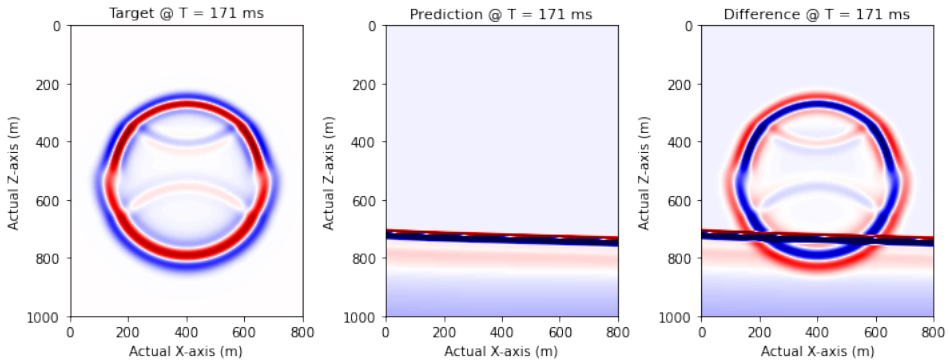

Figure 15: Target (left) and prediction (middle) and their difference (right) for 100 epochs with normalised input values.

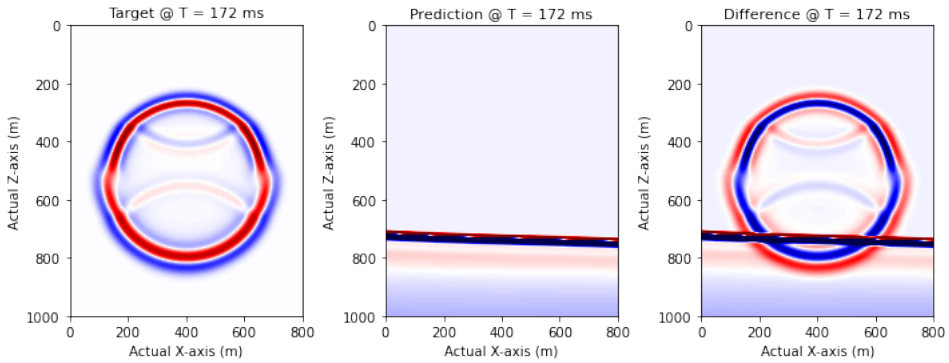

Figure 16: Target (left) and prediction (middle) and their difference (right) for 100 epochs with normalised input values.

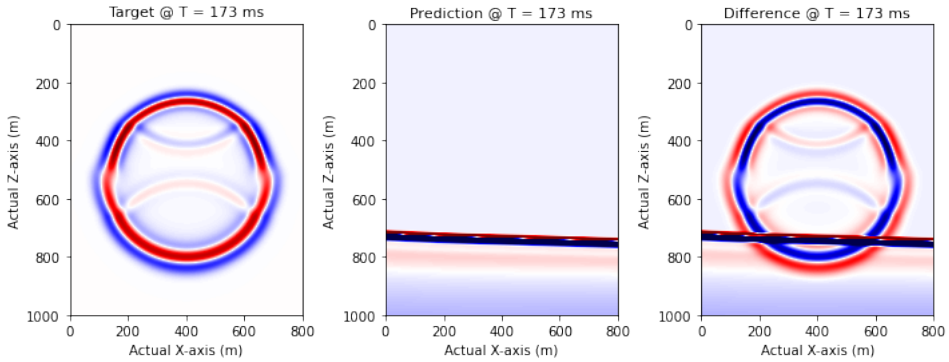

Figure 17: Target (left) and prediction (middle) and their difference (right) for 100 epochs with normalised input values.

Very clearly normalising the values does not work, the data loss is almost stagnant and the prediction of the algorithm is not even close to what it should be. Therefore we have used empirical methods to determine the optimal scaling. After running countless experiments with different scaling values we find that the values in the table below work best.These values greatly improved the accuracy of the network. This can bee seen in figures 18, 19, 20 and 21.

Table 2: Ideal scaling of the input features and the label of this data.

	x	z	t	v	ρ	P
min	30	1.0	0	12.5	10.9	-1
max	40	29.8	23.8	25	13.0	1

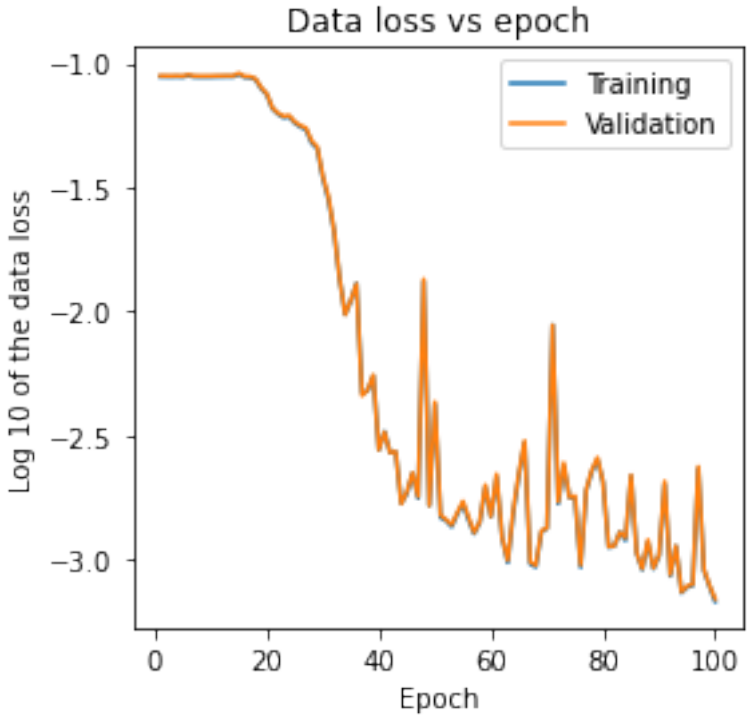

Figure 18: Loss development for optimal scaling for a 100 epochs with PINN disabled.

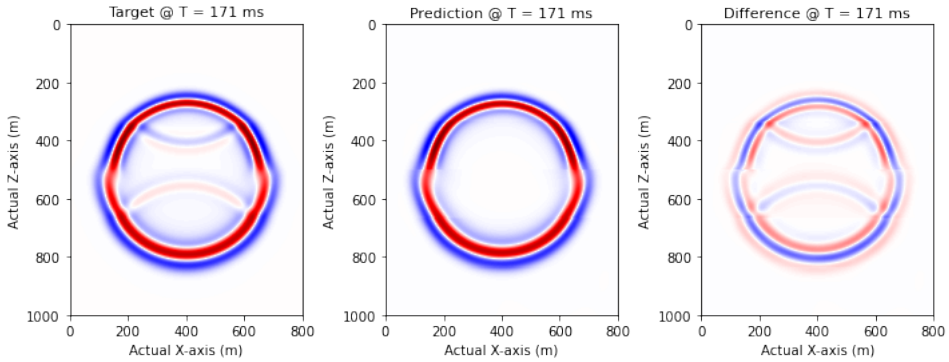

Figure 19: Target (left) and prediction (middle) and their difference (right) for 100 epochs with optimal input values.

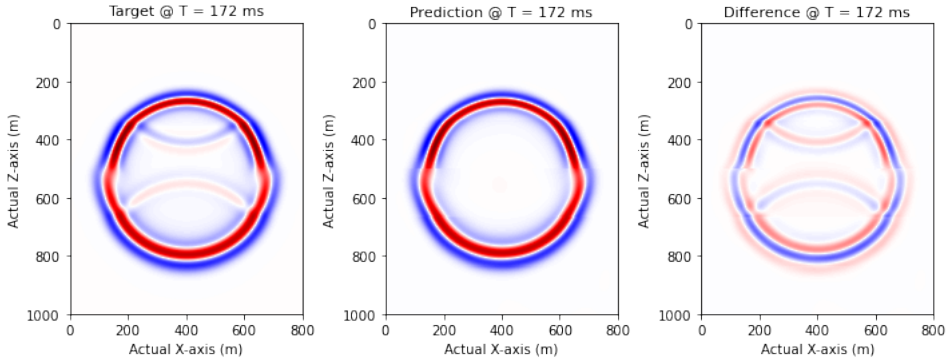

Figure 20: Target (left) and prediction (middle) and their difference (right) for 100 epochs with optimal input values.

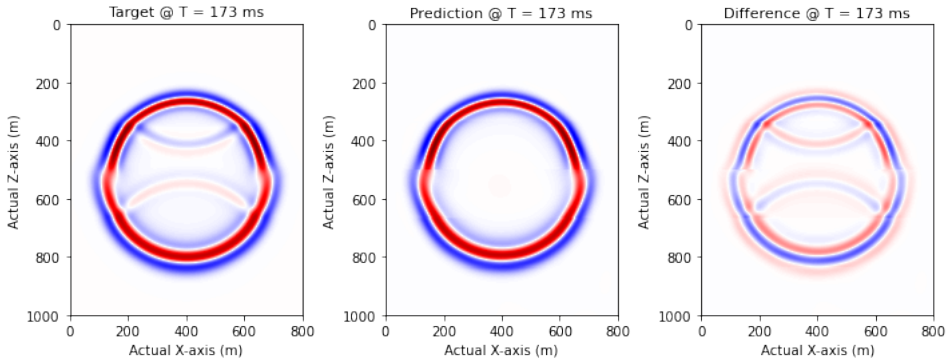

Figure 21: Target (left) and prediction (middle) and their difference (right) for 100 epochs with optimal input values.

4.3. Including the Physics Loss

A problem we ran into during the start of this project was that when the PINN was enabled the network would collapse. All the progress the neural network had made would be lost and the L2 loss function would get a value comparable to it's initial value. This problem arose because the P_{inc} factor in equation 12 was too big, therefore after having trained the network for a bit without the physics loss component the L2 loss was much smaller than the physics loss. Because of this the network only focused on reducing the physics loss component and reverted to the trivial solution of the wave equation. This is shown in figures 22 and 23. After investigating this phenomenon and trying out different values for P_{inc} we have found that it is optimal for P_{inc} to be roughly equal to the size of the L2 loss at the epoch where the PINN is enabled.

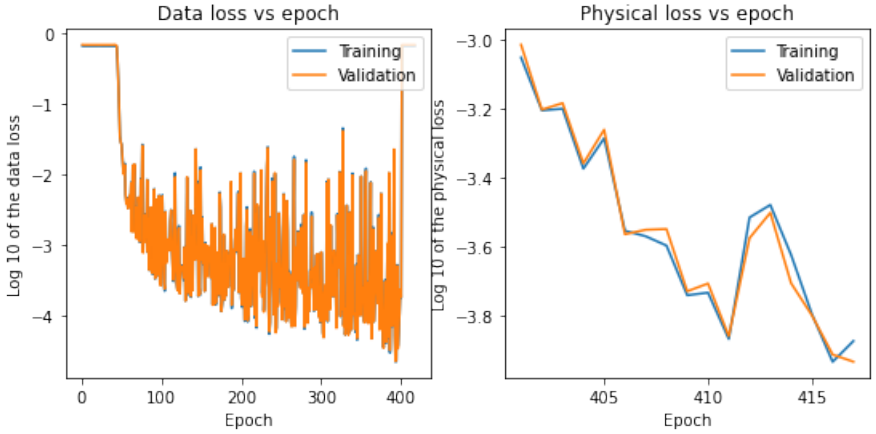

Figure 22: Development of the L2 loss function (left) and the physics loss function (right), PINN is enabled at epoch 400.

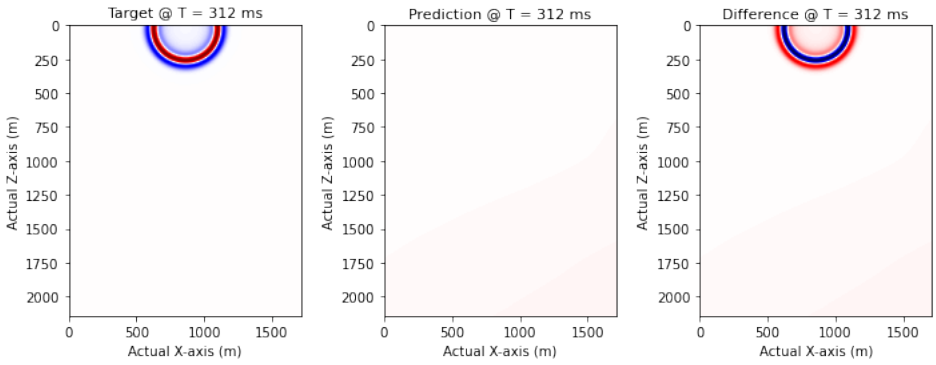

Figure 23: Target (left) and prediction (middle) and their difference (right) for 420 epochs with the last 20 PINN enabled.

4.4. Sampling the physics loss points

As explained in the theory section we are using a new method to sample the physics points. Previously to this method Latin hypercube sampling was used[2]. The results of this change can be found in Figures 24 and 25. As you can see the decrease is more consistent and less erratic, and it also yields a overall better result. This way of sampling also creates a more symmetric wave as can been seen in figures 26 and 27.

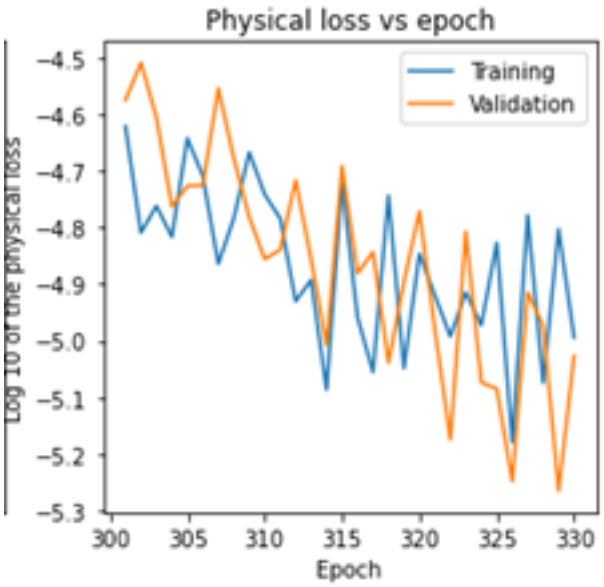

Figure 24: Physics loss development with the Latin hypercube sampling method.

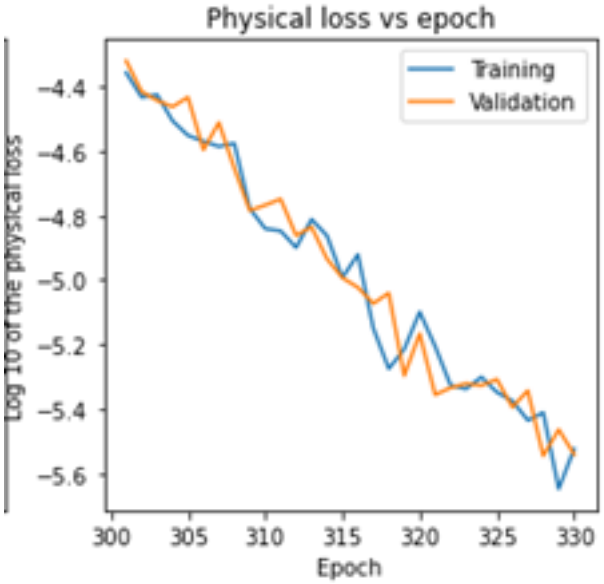

Figure 25: Physics loss development with the new method.

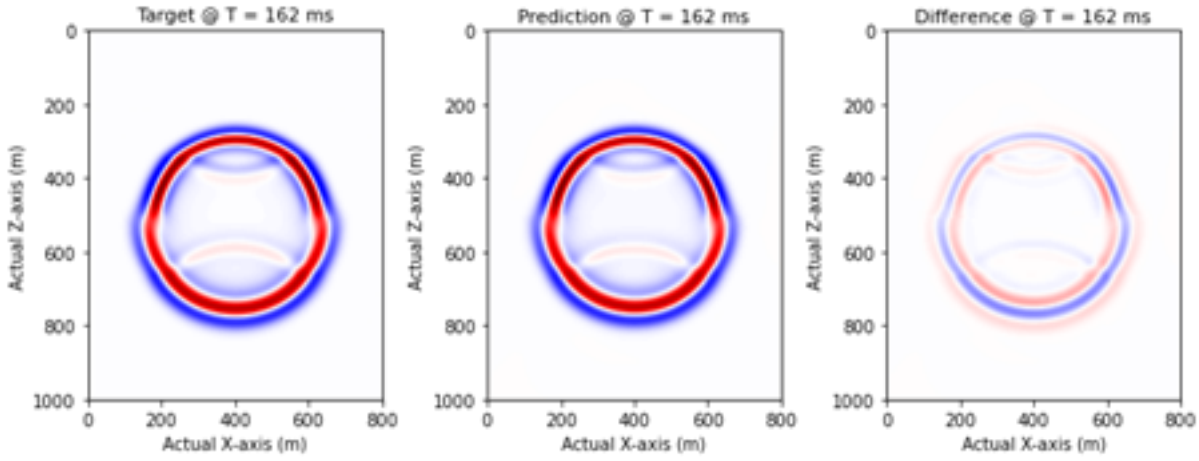

Figure 26: Target (left) and prediction (middle) and their difference (right) with the Latin hypercube sampling method.

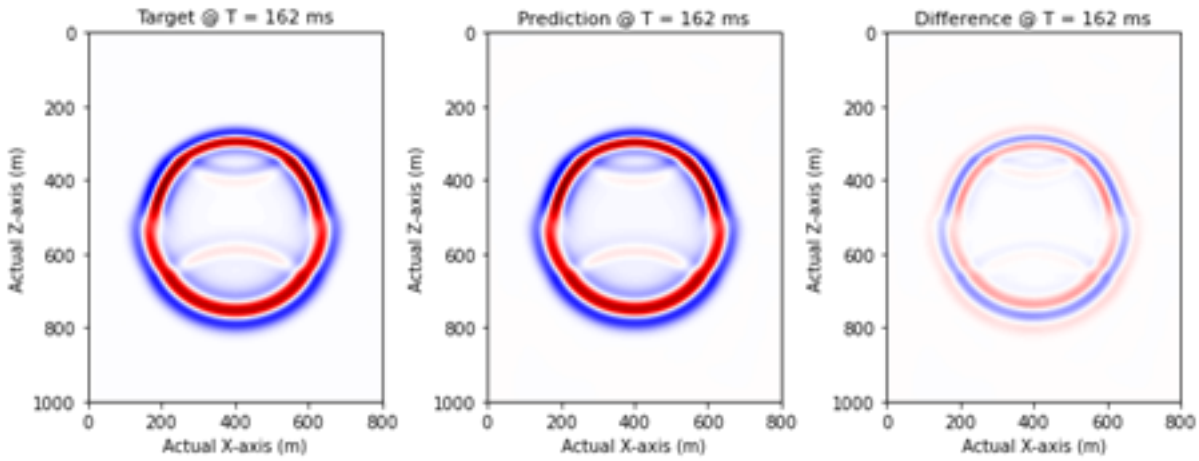

Figure 27: Target (left) and prediction (middle) and their difference (right) with the new method.

4.5. Extensive training times

Finally, we have looked at what happens when we train for a very long period of time, namely 3000 epochs without PINN and 30 with PINN enabled. We find that loss function roughly stagnates but does still decrease even after 3000 epochs, however the physics loss function starts to go up again after a certain amount of epochs, this could be attributed to overfitting. The exact reasons why this happens is uncertain. The loss development and predictions of the network can be found in figures 28, 29, 30 and 31.

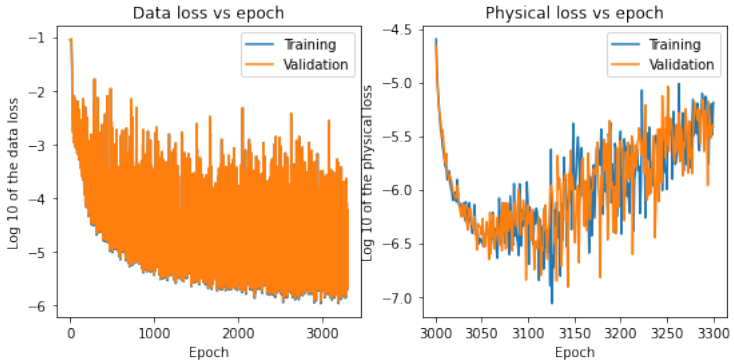

Figure 28: Loss development with 3000 epochs without PINN and 300 with PINN.

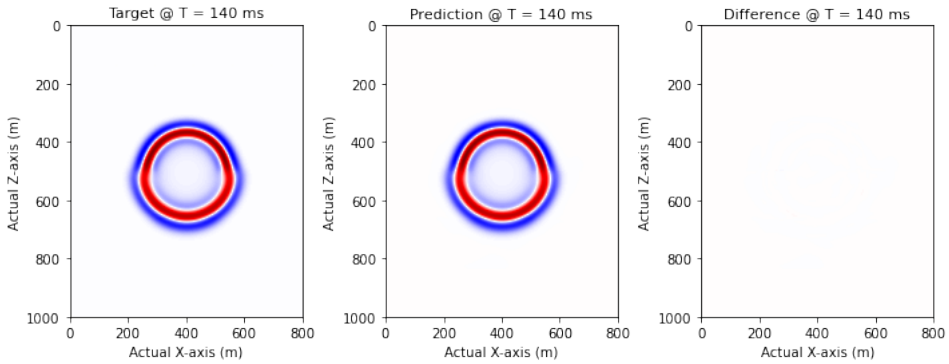

Figure 29: Target (left) and prediction (middle) and their difference (right) of the network.

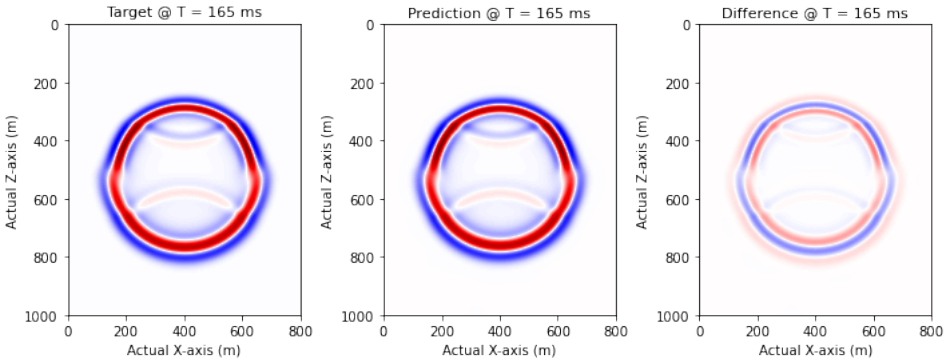

Figure 30: Target (left) and prediction (middle) and their difference (right) of the network.

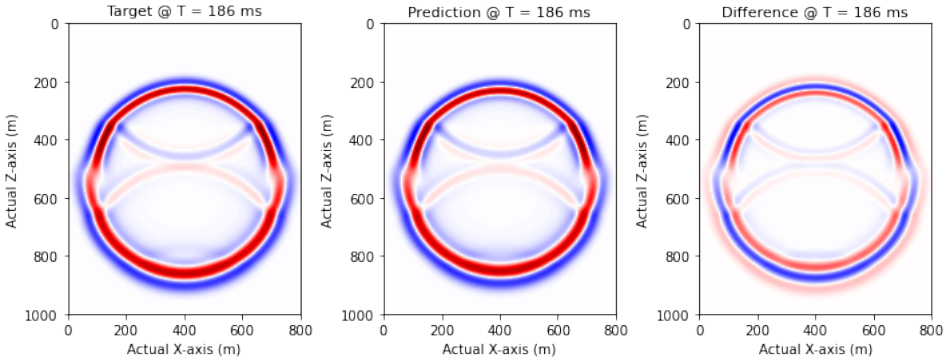

Figure 31: Target (left) and prediction (middle) and their difference (right) of the network.

5. Conclusion

We have analysed the the behaviour of a physics informed neural network and its competence in predicting a wave in a non-homogeneous medium. We have looked at many aspects of the network and have achieved significantly better results than we had in the past. During the project it has become clear that neural networks are very capable of predicting the behaviour of the wave within and in between the timeframe of the given training data. The network is also somewhat capable of predicting the past and future development of the wave. The predictive capabilities of the network stagnate relatively quick when you train for extensive amounts of time.

During this project we have been focused on reducing the loss function in the shortest possible time, this has made us extremely vulnerable to overfitting. In literature it is shown that the PINN allows the network to find the underlying generalisation. It is uncertain whether our network achieved this as the network in literature is much better at predicting the behaviour of the wave outside of the range of the training data than our network.

For future research we would recommend to look at a network structure with much more neurons per layers, this is also one of the main differences between our network and networks found in literature with better results. We should seriously ask ourselves, is it possible to capture a generalisation of this complex physical phenomenon in only 80 neurons? Using the 1024 neurons like is done in literature, is out of the scope of this project, as the training time required to train such a network changes from hours to weeks. Another thing that more research should be done on is that the network accuracy has a negative correlation with the time parameter, why does the network prefer earlier timestamps? Is this part of the nature of the network or is it because of a underlying systemic error? Addressing these problems may yield better results and improve the predictions of the network outside of the given timeframe.

References

[1] Ben Moseley, Andrew Markham, and Tarje Nissen-Meyer. Solving the wave equation with physics-informed deep learning. *arXiv preprint arXiv:2006.11894*, 2020.

[2] Jesse Buijs. Investigating physics informed neural networks for acoustic wave modeling. 2021.

[3] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[4] A. Glassner. *Deep Learning: A Visual Approach*. No Starch Press, 2021.

[5] IBM Cloud Education. Neural networks. `https://www.ibm.com/cloud/learn/neural-networks`. Accessed on 2021-05-11.

[6] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018.

[7] Donald Bertucci. Backprop explainer. `https://xnought.github.io/backprop-explainer/`. Accessed on 2021-11-11.

[8] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

Appendix

To get a hands on experience on how machine learning works I would recommend checking out this fun tool tensorflow made: https://playground.tensorflow.org/

Acknowledgements

First I would like to thank my supervisor, Dr. Eric Verschuur, who has guided me through this project and has assisted me with writing this thesis.

Next I would like to thank Drs. Jan van de Mortel for his help with the PINN and finite difference model during the start up of this project and his support whenever I ran into problems.

I would also like to thank Dennis Schoonhoven, Dr. Shan Qu and Drs. Jan van de Mortel for developing the PINN to the current version.