FYS-STK3155/4155
Project 3
Modeling Heat Conduction And Its Stability: Numerical and Data-Driven Approaches

Ben René Bjørsvik, Max Dahl, Egil Rolstad, and Asgeir Kråkenes

December 2024

Full project material available in our GitHub repository.

Contents

1 Abstract

Accurate and efficient solutions to Partial differential equations (PDEs) are critical in fields such as physics, engineering and computational sciences. Recently, neural networks have attracted attention for their flexibility and ability to model complex phenomena. In this paper, we compare a classic explicit finite-difference scheme with Feed Forward Neural Networks (FFNNs) and Recurrent Neural Networks (RNNs) for solving a one-dimensional diffusion equation. Although both neural network models closely approximate the analytical solution, the explicit scheme is superior in performance - assuming that a certain stability criterion is satisfied. However, if the stability criterion is violated, the explicit scheme collapses and the neural network methods are the sole effective solutions. In other words, which models performs best depends on the problem at hand.

2 Introduction

The rapid development and advancement of neural networks, along with their dual inspiration from and contributions to the field of physics, were recognized with the awarding of this year's Nobel Prize in Physics [1, 2, 3]. One of the laureates' (John Hopfield) work drew inspiration from how models in physics describe the interaction of small components within a system and how they shape the system's overall behavior [3]. The result is algorithms that is today affecting millions of lives.

Nevertheless, this "relationship" between physics and neural networks is in fact bidirectional: while physics has inspired these neural network models, the same models are proving to be powerful tools to model and advance physics and other scientific disciplines [4]. As such, many recent advances in our understanding of the physical world are driven by large-scale computational modeling and data analysis [4]. More specifically, in computational physics, neural networks are employed as function approximators while reducing computational requirements. Here, these network architectures effectively replicate physical models and enable the analysis of larger systems at higher resolutions. Examples are areas within quantum-mechanics, data analysis of particle physics, as well as the employment of such methods to explore phase transitions and thermodynamic properties [4].

In this project, we aim to give a demonstration of the utility of neural networks for modeling such a physical process evolving across time and space. We assess using neural networks for solving partial differential equations (PDEs): a multi-variable function and one or more of this function's derivatives. PDEs are central to a vast variety of applications, including the modeling of economics and populations in biology, in addition to describing a wide range of physical phenomena, such as the problem we are looking at in this project.

Specifically, we focus on the solution of the diffusion equation in one dimension. This is a PDE that can describe the temperature gradient in a one-dimensional rod of a given length as a function of time and space with some assumptions. Thus, it can serve as a basic model for the heat conduction in the rod. After introducing our problem, we determine the analytical solution in section 3. Consequently, we approach the solution of the diffusion equation using three methods: a standard explicit scheme and two different neural networks, FFNN and RNN. In contrast, the explicit scheme provides a classical numerical solution to the problem, while the RNN model, which is particularly well-suited for modeling sequential data, offers a data-driven approach.

By comparing these methods, we find that the numerical method proves to be superior when a stability criterion – introduced in section 3.1.2 – is satisfied, and that the neural networks are the sole effective solutions when this is not the case. Furthermore, we also find that the best-performing RNN model is capable of converging faster than the one of the FFNN. These findings are presented in section 4. Then, we discuss these results, including the limitations that our numerical model faces. Lastly, we discuss the advantages neural networks can have for pattern recognition in scenarios of numerical limitations and data of higher dimensions, upon our final concluding remarks in section 5.

3 Theory and methods

In this section we explain the methods we use to predict the diffusion equation and their underlying theory. We start by introducing the diffusion equation. We present our particular case of the diffusion equation, derive it's analytical solution, and show how the explicit scheme is used to find numerical solutions. Then, we briefly introduce feedforward neural networks (FFNNs) and recurrent neural networks (RNNs), and show how these are used to solve the diffusion equation.

3.1 The diffusion equation

Partial differential equations are frequently used in physical problems to formalize, quantize and explain the behavior of e.g. materials, gasses, and fluids across both space and time. In this project, we are interested in modeling the evolution of temperature on a one-dimensional rod over time. The temperature function is formalized as $u(x,t)$, where x is the position on the rod and t the time. To find the function $u(x,t)$, we consider the *diffusion equation*:

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t}, t > 0, x \in [0, L] \tag{1}$$

Note that the diffusion of a substance in a given medium usually requires a diffusion constant D, say, particular for the problem at hand. For the equation above, and for the remaining part of this report, we consider the dimensionless case, in which no diffusion constant D is needed. For details on this see [5] page 304.

We consider a rod of length $L = 1$ with initial conditions at $t = 0$,

$$u(x, 0) = \sin(\pi x) \quad 0 < x < L, \tag{2}$$

and boundary conditions

$$u(0, t) = 0, \quad u(L, t) = 0 \qquad t \geq 0. \tag{3}$$

We will now see how this equation may be solved.

3.1.1 Analytical solution

A first attempt is to find the analytical solution to the equation. In this section, as in the next, we heavily rely on the theory presented in [5].

We start by assuming the solutions are of the form

$$u(x, t) = F(x)G(t),$$

i.e. the variables are separable. Inserted into the diffusion equation 3.1 gives

$$\begin{aligned} \frac{\partial^2 F(x)G(t)}{\partial x^2} &= \frac{\partial F(x)G(t)}{\partial t} \\ \implies F''(x)G(t) &= F(x)G'(t) \\ \implies \frac{F''(x)}{F(x)} &= \frac{G'(t)}{G(t)}. \end{aligned}$$

This equation holds for all x and t. The diffusion equation assumes conservation of energy, meaning that for some value (x, t), the left and right hand side both should equal to some constant, say $-\lambda^2$. We then have the following two differential equations

$$F''(x) + \lambda^2 F(x) = 0, \qquad G'(t) = -\lambda^2 G(t),$$

which have the general solutions

$$F(x) = A\sin(\lambda x) + B\cos(\lambda x), \qquad G(t) = Ce^{-\lambda^2 t}$$

for some constants A, B, C.
From the boundary conditions in 3, we have that for any $t \geq 0$,

$$(A\sin(0) + B\cos(0))Ce^{-\lambda^2 t} = 0, \qquad (A\sin(\lambda L) + B\cos(\lambda L))Ce^{-\lambda^2 t} = 0.$$

From the first equation, we see that B must be 0. From the second equation, we have that

$$\sin(\lambda L) = 0 \implies \lambda = n\pi/L$$

for all integers $n \geq 0$. Then, a solution for $u(x,t)$ is

$$u_n(x,t) = A_n \sin(n\pi x/L) e^{-n^2\pi^2 t/L^2}.$$

We see that this solution depends on n. However, using our initial condition we have that

$$\begin{aligned} u_n(x,0) = \sin(\pi x) &= A_n \sin(n\pi x/L) e^{-n^2\pi^2 \cdot 0/L^2} \\ \implies \sin(\pi x) &= A_n \sin(n\pi x/L) \overset{L=1}{=} A_n \sin(n\pi x). \end{aligned}$$

We see that the initial condition is satisfied for $n = 1$ and $A_1 = 1$. We then obtain the solution

$$u(x,t) = \sin(\pi x) e^{\pi^2 t}. \tag{4}$$

3.1.2 Explicit scheme solution

We now show how a solution to the diffusion equation may be obtained numerically, using an equal-step method. In short, we use our inital condition and boundary conditions to approximate the rod temperature in a given timepoint based on the temporally previous temperature and it's spatially adjacent temperatures. As for the analytical solution in the previous section, we rely on the theory provided in [5].
When solving the equation numerically, our space and time dimensions x and t must be discretized. Let the step length of x be

$$\Delta x = \frac{1}{n+1}$$

for some non-negative integer n, which sets our spatial resolution for the discretized solution. Similarly, we pick some suitable time step Δt relative to Δx (more on this below). Now, the position after i steps and time at time at time-step j are given by

$$\begin{aligned} t_j &= j\Delta t \qquad j \geq 0, \\ x_i &= i\Delta x \quad 0 \leq i \leq n+1. \end{aligned}$$

Approximating the derivatives gives

$$\frac{\partial u}{\partial t} \approx \frac{u(x, t+\Delta t) - u(x,t)}{\Delta t} = \frac{u(x_i, t_j + \delta t) - u(x_i, t_j)}{\Delta t}$$

with local approximation error $\mathcal{O}(\Delta t)$, and similarly

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(x_i + \Delta x, t) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)}{\Delta x^2}$$

with a local approximation error $\mathcal{O}(\Delta x^2)$. Simplifying these equations gives

$$\frac{\partial u}{\partial t} \approx \frac{u_{i,j+1} - u_{i,j}}{\Delta t}, \quad \frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}.$$

We now have a dicretized version of the diffusion equation:

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}.$$

Defining $\alpha = \Delta x / \Delta x^2$ gives us the following explicit scheme:

$$u_{i,j+1} = \alpha u_{i-1,j} + (1 - 2\alpha)u_{i,j} + \alpha u_{i+1,j}.$$

Our initial condition tells us that $u_{i,0} = \sin(\pi x_i)$, which gives the next step at position i

$$\begin{aligned} u_{i,1} &= \alpha u_{i-1,0} + (1 - 2\alpha)u_{i,0} + \alpha u_{i+1,0} \\ &= \alpha \sin(\pi x_{i-1}) + (1 - 2\alpha)\sin(\pi x_i) + \alpha \sin(x_{i+1}) \end{aligned}$$

To find the next point $u_{i,2}$ using $u_{i,1}$, we additionally use the boundary conditions $u_{0,t} = u_{1,t} = 0$.

We mentioned that Δt have to be picked relative to Δx. Without going into the details here, one can show that the explicit scheme yields stable solutions only if the *stability criterion* is met, namely when

$$\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}.$$

Intuitively, we may think of this condition requiring a certain proportion of the number of steps in time versus space. For an increasing number of timesteps, the stability of solutions requires a certain increase in steps in space. This criterion comes from an analysis of numerical stability, as explained in [5] page 307-308.

3.2 Neural networks

Neural networks are computational models inspired by the structure and function of the human brain. They consist of layers of interconnected nodes, where each connection is assigned a weight that is adjusted during training. Neural networks have been applied to a wide range of problems, including classification, regression, and the numerical approximation of solutions to differential equations.

3.2.1 FFNN

To solve the PDE described in 3.1, we implement a Feed Forward Neural Network (FFNN), described in detail in [2]. We wish to find a function $u(x,t)$ satisfying the PDE conditions. To do this, we utilize a trial solution of the form

$$u_t(x,t) = (1-t)\sin(\pi x) + x(1-x)tN(x,t;P),$$

where $N(x, t; P)$ is the output from an FFNN defined by the parameters P (weights and biases) [6]. In this way, the trial solution automatically satisfies the initial conditions of the PDE. Our cost function is defined as

$$C = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{\partial u_t(x_i, t_i)}{\partial t} - \frac{\partial^2 u_t(x_i, t_i)}{\partial x^2} \right)^2,$$

where the partial derivatives of the trial solution are found using automatic differentiation in Tensorflow. The sum is taken over the points (x_i, t_i), which is a discretization of the feature space. The FFNN is then trained using backpropagation with ADAM as the optimizing algorithm.

Our FFNN model is implemented using Tensorflow/Keras [7]. The implementation allows for easy tuning of hyperparameters such as number of hidden layers, number of nodes, activation function and learning rate.

3.2.2 RNN

Recurrent neural networks (RNNs) are a type of neural network designed to process sequential data. This makes them well-suited for solving time-dependent PDEs, as the temporal evolution of the solution can be modeled naturally through sequential layers corresponding to time steps.

In this approach, each time step corresponds to a layer in the RNN. The input to the network at each layer consists of the spatial coordinate x and the output at the previous time step. The RNN produces an output at each time step, which is then compared to the analytical solution of the PDE (4), and the error is propagated backward through the network to adjust the weights [8].

Mathematically we can define a single hidden layer RNN as

$$z_h^{(t)} = W_{hx} x^{(t)} + W_{hh} h^{(t-1)} + b_h,$$

$$h^{(t)} = \sigma_h(z_h^{(t)}),$$

[9] where at each time step t, the hidden state $h^{(t)}$ is updated using the input $x^{(t)}$ and the previous hidden state $h^{(t-1)}$. W_{hx} and W_{hh} are weight matrices mapping the input to the hidden state and the previous hidden state to the current hidden state. Furthermore, b_h is a bias term, and σ_h is an activation function [2].

The output at time t is computed from the hidden state:

$$z_y^{(t)} = W_{yh} h^{(t)} + b_y,$$

$$y^{(t)} = \sigma_y(z_y^{(t)}),$$

[9] where W_{yh} and b_y are the output weight matrix and bias, and σ_y is the output activation function.

The RNN is trained by minimizing the total loss over all time steps, using Backpropagation through time [10]:

$$L = \sum_{t=1}^{T} L^{(t)},$$

The gradient of L with respect to the recurrent weights W_{hh} is computed as:

$$\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{T} \frac{\partial L^{(t)}}{\partial y^{(t)}} \frac{\partial y^{(t)}}{\partial h^{(t)}} \sum_{k=1}^{t} \frac{\partial h^{(t)}}{\partial h^{(k)}} \frac{\partial h^{(k)}}{\partial W_{hh}}$$

where $\frac{\partial h^{(t)}}{\partial h^{(k)}}$ involves repeated multiplication of Jacobians:

$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial h^{(i)}}{\partial h^{(i-1)}}.$$

[10] We implement the RNN using `SimpleRNN` layers in Tensorflow/Keras. The model is trained by minimizing the mean square error between the trial solution and the analytical solution, again utilizing the ADAM optimizer.

4 Results

4.1 Explicit scheme

We evaluated the explicit scheme algorithm for two scenarios, one with $\Delta x = 1/10, \Delta t = 1/600$ and another with $\Delta x = 1/100, \Delta t = 1/6000$. Both of these satisfy the stability criterion, and gave the solution surfaces depicted in figure 1.

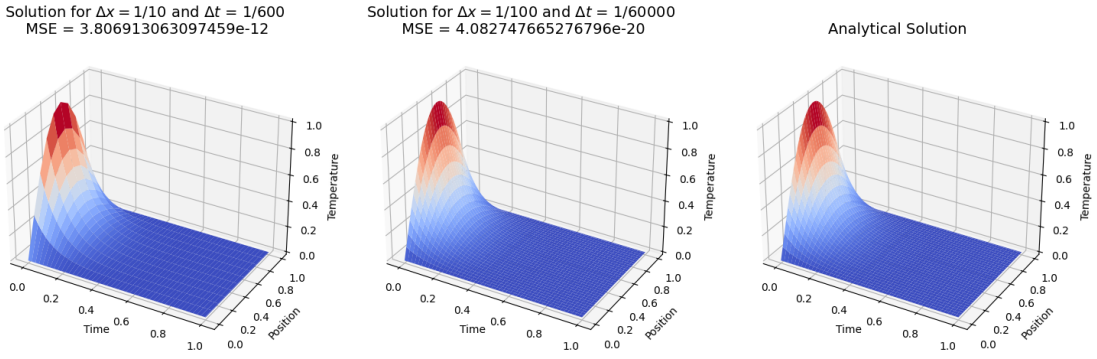

Figure 1: *Explicit scheme solutions for two different combinations of time and space steps. The reported MSE values are calculated relative to the (true) analytical solution for the corresponding number of time steps.*

To evaluate how the temperature evolves over time, we also picked two timesteps for each of the time and space step combinations, reported in figure 2 as slices in the surfaces from figure 1.

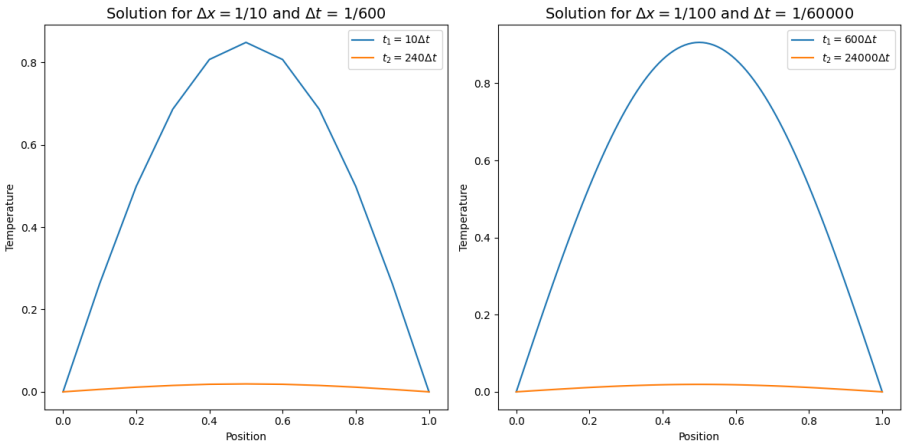

Figure 2: *The temperature distribution of the rod at two timepoints for each of the time and space step combinations.*

We can tell that both solutions lie very close to the true surface, as indicated by the very low MSE scores. From the slices, we can also tell how the different number of of steps in space, i.e. the resolution of the rod, makes the solution for $\Delta x = 1/10$ more pointy than the smooth solution when $\Delta t = 1/100$.
So far, it seems like the explicit scheme works nearly perfect to solve the diffusion equation. However, we also assessed how it performs outside the stability criterion $\Delta t/\Delta x^2 \leq 1/2$, illustrated in figure 3.

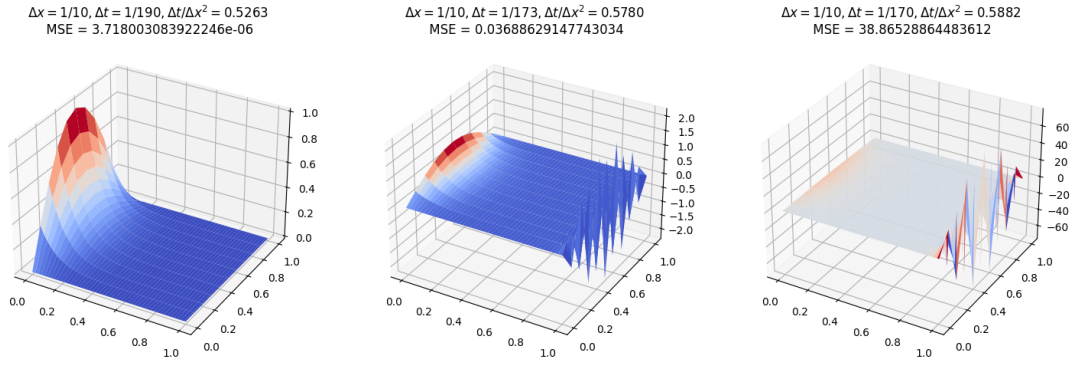

Figure 3: *Solutions of the explicit scheme for space and timesteps violating the stability criterion.*

We clearly see how the explicit scheme collapses for increasing violation of the stability criterion. As before, the MSE reported in the plot is the error relative to the analytical solution. The MSEs increases correspondingly to the collapsing surfaces, as we would expect.

4.2 Neural Networks

From fig. 5, we see that both the FFNN and RNN model approximate the analytical solution to indistinguishability. This is similar to the results observed for the explicit scheme with $\Delta x = \frac{1}{100}$ and $\Delta t = \frac{1}{60\,000}$ (section 4.1), suggesting that there is not much to gain from using more complex methods than an explicit scheme in this case.

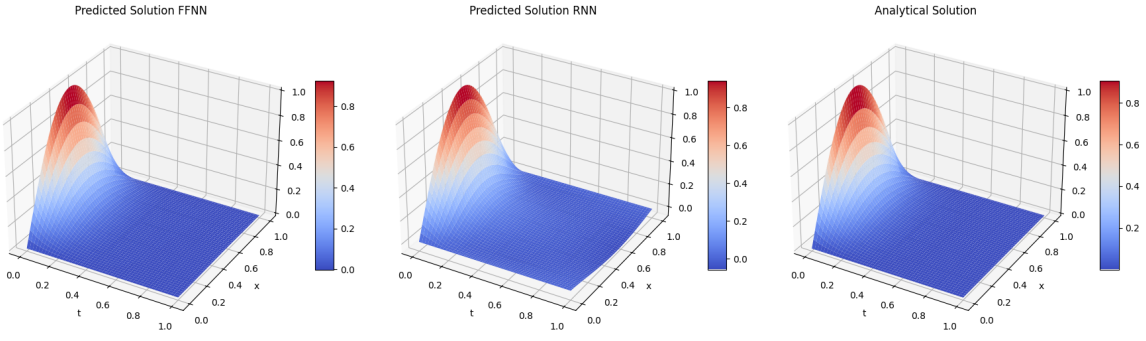

Figure 4: *Predicted surface for FFNN model with 2 hidden layers, 10 nodes per layer, sigmoid activation function and learning rate of 0.01. Predicted surface for RNN model with 3 hidden layers, 20 nodes per layer,* tanh *activation function and learning rate of 0.01. Shown together with a plot of the analytical solution for* $t \in [0, 1]$.

To investigate how the choice of hyperparameters influenced our results, we trained many different FFNN and RNN models. For each combination of hyperparameters, we found the post-training loss. The result from this parameter tuning for the FFNN and RNN models can be found in table 1 and table 2 respectively. We found that for the FFNN, the best model was obtained with 3 hidden layers, 20 nodes per layer, sigmoid activation function and a learning rate of 0.1, yielding a loss of 0.0017. For the RNN, the optimal parameters were 3 hidden layers, 20 nodes per layer, tanh activation function and a learning rate of 0.01, resulting in a loss of $6.18 \cdot 10^{-7}$. These losses are not comparable, as the FFNN and RNN utilize different loss functions. If we consider the RNN loss function, namely the MSE between predicted output and the analytical solution, the best FFNN corresponds to an MSE of $8.12 \cdot 10^{-5}$. Hence, while the best RNN model slightly outperforms the best FFNN model, the difference is likely just a coincidence.

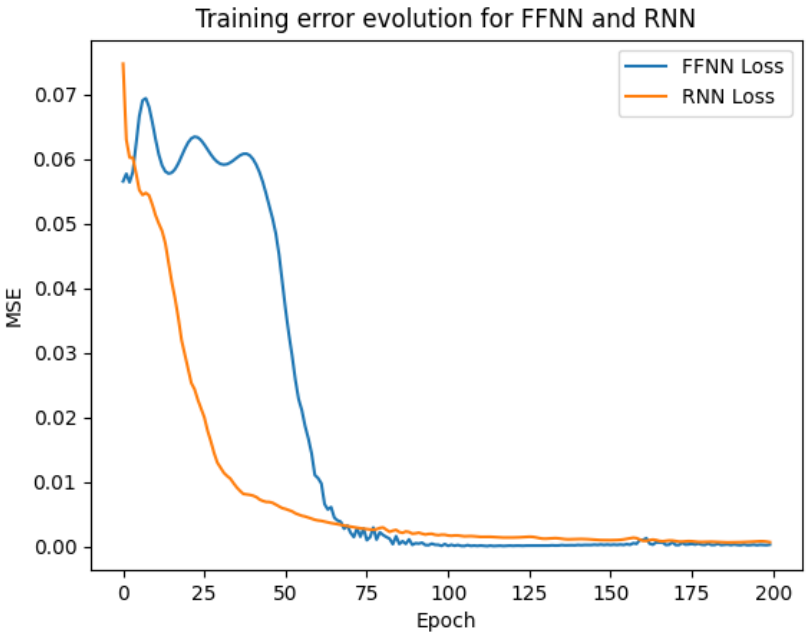

Figure 5: *Training loss evolution over 200 epochs for the best FFNN and RNN models. The FFNN model has 3 hidden layers, 20 nodes per layer, sigmoid as the activation function, and Adam as the optimizer (learning rate: 0.1). The RNN model has 3 hidden layers, 20 nodes per layer,* tanh *as the activation function, and Adam as the optimizer (learning rate: 0.01). Loss is measured as Mean Squared Error (MSE). Both models are trained on a 10x10 grid for the diffusion equation with analytical solutions for* $t \in [0, 1]$

The plot illustrates the training loss over 200 epochs for the best-performing FFNN and RNN models as described above in the analysis for the plot of the predicted surfaces.

The RNN demonstrates significantly faster convergence during the early epochs, stabilizing at a low error earlier may due to its ability to effectively model temporal dependencies. In contrast, the FFNN converges more slowly but eventually achieves a comparable low error with sufficient training. While the RNN converges faster and performs slightly better, as seen in the tables in the Appendix, both models show strong results, highlighting the value of good architecture and hyperparameter selection.

4.3 Comparing neural networks and explicit scheme

We now want to compare the above mentioned methods. We saw that the explicit scheme had MSEs of order 10^{-12} and 10^{-20}, depending on the grid size, whereas the FFNN and RNN models had losses of order 10^{-7} and 10^{-5}. Thus, it may seem that the explicit scheme outperforms both the neural network models up to fifteen-fold - at least within the stability criterion.
When solving the diffusion equation numerically, an ideal solution is flexible in the space and time scale, i.e. the number of steps in space and time. This way, the same solution may be used for different problems requiring different spatial and temporal resolutions. We have seen that the stability criterion for the explicit scheme greatly restricts valid spatial and temporal resolution relative to each other. Violating the stability criterion gives nonsensical explicit scheme solutions, making the explicit scheme not a very flexible solution. In figure 6 we compare the explicit scheme with the two neural network methods in cases where the stability criterion is and is not satisfied. The neural networks were trained using the optimal hyperparameters found in the previous section.

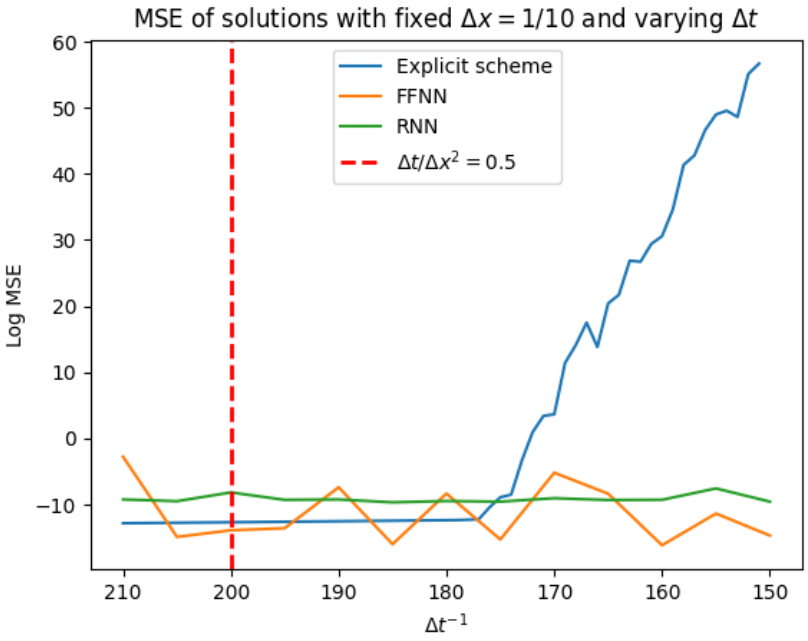

Figure 6: *Comparing the explicit scheme with the neural network methods for a fixed number of space steps and varying number of time steps. Everything to the right of the red dotted line, i.e. the stability criterion, violates the criterion. The (log) MSE is still the error relative to the analytical solution for the corresponding grid size.*

From figure 6, we can tell that the explicit scheme performs on average best up to a given point, approximately at $\Delta t^{-1} = 175$, after which its MSE explodes. Meanwhile, the neural network methods seem to be stable across the Δts, with the RNN being the most stable. We note that we sample a fewer number of neural network solutions, due to computational constraints, which may contribute to some of the large jumps, especially for the FFNN.

5 Discussion and conclusions

When the stability criterion is satisfied, the explicit finite difference scheme offers accurate results at comparatively low computational cost. Its downside is the strict interdependence between temporal and spatial resolution. If one desires a fine spatial resolution, which require smaller spatial steps, the explicit scheme also enforces increasing the number of time steps, resulting in unwanted computational cost.

In contrast, the neural network approaches (FFNN and RNN) allow more freedom in setting spatial and temporal discretizations. The FFNN only needs the differential equation itself, making it suitable for cases where an analytical solution is not available. Its performance can be somewhat inconsistent, but it avoids reliance on known solutions. The RNN tends to model temporal dependencies more directly, yet it typically requires an analytical reference solution as seen, which may limit its applicability in many practical scenarios.

The preferable choice of method to numerically solve the diffusion equation seems to be heavily dependent on the problem at hand. When the stability criterion of the explicit scheme is satisfied, the larger neural network models may be considered superfluous, wheres outside the stability criterion the explicit scheme collapses. While not tested here, it would also be interesting to compare the methods for higher dimensional problems.

6 Bibliography

[1] *The Nobel Prize in Physics 2024 - Press Release*. The Nobel Prize. 2024. URL: https://www.nobelprize.org/prizes/physics/2024/press-release/ (visited on 11/08/2024).

[2] B. Bjørsvik et al. "Simplicity Versus Complexity: Traditional Statistical Methods and Neural Networks for Regression and Classification". In: *FYS-STK3155/4155 Fall 2024 Project 2* (Nov. 2024).

[3] The Nobel Prize in Physics 2024 - Laureats. *John J. Hopfield – Facts*. Accessed: 2024-12-15. 2024. URL: https://www.nobelprize.org/prizes/physics/2024/hopfield/facts/.

[4] The Nobel Committee for Physics. *Scientific Background to the Nobel Prize in Physics 2024: "For Foundational Discoveries and Inventions That Enable Machine Learning with Artificial Neural Networks"*. Accessed: 2024-12-15. The Royal Swedish Academy of Sciences. 2024. URL: https://www.nobelprize.org/uploads/2024/11/advanced-physicsprize2024-3.pdf.

[5] Morten Hjort-Jensen. *Computational Physics Lecture Notes Fall 2015*. 2015. Chap. 10.2 Diffusion equation. URL: https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf.

[6] Morten Hjorth-Jensen and Kristine Baluka Hein. *Week 43: Deep Learning: Constructing a Neural Network code and solving differential equations*. 2024. Chap. Solving differential equations with Deep Learning. URL: https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week43.ipynb.

[7] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[8] Morten Hjorth-Jensen. *Week 45, Convolutional Neural Networks (CCNs) and Recurrent Neural Networks (RNNs)*. 2024. Chap. Solving differential equations with RNNs. URL: https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week45.ipynb.

[9] Morten Hjorth-Jensen. *Week 45, Convolutional Neural Networks (CCNs) and Recurrent Neural Networks (RNNs)*. 2024. Chap. RNNs in more detail, part 7. URL: https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week45.ipynb.

[10] Morten Hjorth-Jensen. *Week 45, Convolutional Neural Networks (CCNs) and Recurrent Neural Networks (RNNs)*. 2024. Chap. Backpropagation through time. URL: https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week45.ipynb.

7 Appendix

7.1 Hyperparameter Tuning

Table 1: Results from hyperparameter tuning of FFNN model

n_hidden	n_nodes	activation	learning_rate	loss
3	20	sigmoid	0.100000	0.001703
3	10	sigmoid	0.100000	0.004220
5	5	sigmoid	0.100000	0.004857
3	20	tanh	0.010000	0.005473
5	20	tanh	0.010000	0.006349
1	20	tanh	0.100000	0.012499
5	10	sigmoid	0.010000	0.019849
3	20	sigmoid	0.010000	0.020581
5	10	tanh	0.010000	0.027348
5	20	sigmoid	0.010000	0.028002
1	10	tanh	0.100000	0.044840
1	20	sigmoid	0.100000	0.047249
3	10	tanh	0.010000	0.062806
3	20	tanh	0.100000	0.082844
1	20	tanh	0.010000	0.086914
3	5	tanh	0.100000	0.091686
1	10	sigmoid	0.100000	0.128987
3	5	tanh	0.010000	0.141588
1	10	tanh	0.010000	0.263329
5	5	tanh	0.010000	0.361012
1	5	tanh	0.100000	0.362485
1	20	sigmoid	0.010000	0.435832
1	5	sigmoid	0.100000	0.525936
3	5	sigmoid	0.100000	0.669466

Table 2: Results from hyperparameter tuning of RNN model

n_hidden	n_nodes	activation	learning_rate	loss
3	20	tanh	0.01	6.184×10^{-7}
3	20	relu	0.01	1.901×10^{-6}
5	20	relu	0.01	4.515×10^{-6}
3	10	tanh	0.01	5.532×10^{-6}
1	20	tanh	0.1	5.848×10^{-6}
1	20	relu	0.01	7.920×10^{-6}
5	20	relu	0.001	9.222×10^{-6}
1	10	tanh	0.1	1.158×10^{-5}
1	20	relu	0.1	1.167×10^{-5}
3	10	relu	0.001	1.557×10^{-5}
1	10	sigmoid	0.1	1.719×10^{-5}
5	20	tanh	0.01	1.761×10^{-5}
3	20	sigmoid	0.1	2.096×10^{-5}
1	5	tanh	0.1	2.288×10^{-5}
5	10	tanh	0.01	2.461×10^{-5}
5	10	relu	0.01	3.374×10^{-5}
3	5	relu	0.01	4.047×10^{-5}
3	10	sigmoid	0.1	4.532×10^{-5}
3	20	relu	0.001	4.582×10^{-5}
1	20	sigmoid	0.1	4.627×10^{-5}
5	5	relu	0.001	4.704×10^{-5}
1	5	sigmoid	0.1	5.058×10^{-5}
1	20	relu	0.001	6.782×10^{-5}
3	5	tanh	0.01	6.955×10^{-5}