BAB VI
PENUTUP

A. Kesimpulan

Berdasarkan hasil penelitian dan diskusi, dapat disimpulkan bahwa:

1. ANN untuk menghitung PDEs telah berhasil dikembangkan beserta FDM dan jawaban eksak sebagai metode pembanding.
2. ANN mampu menghasilkan hasil kompetitif dibandingkan FDM, yakni hingga 1.5 kali lebih akurat, sehingga benar bahwa ANN dapat menjadi metode alternatif untuk menyelesaikan PDEs. Namun, waktu eksekusi menjadi kelemahan utama ANN.

B. Saran

Dengan mempertimbangkan hasil dari penelitian sekarang, penelitian kedepan disarankan untuk:

1. Melakukan eksplorasi hyperparameter lebih jauh pada kasus dua dimensi untuk mendapatkan hasil ANN yang lebih optimal.
2. Mendalami cara kerja loop agar dapat menghasilkan program yang lebih cepat dan hasil yang lebih akurat.
3. Mempertimbangkan penggunaan GPU untuk mempercepat proses perhitungan ANN agar eksplorasi hyperparameter dapat dilakukan lebih jauh.

DAFTAR PUSTAKA

[1] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," Jun. 2014, doi: 10.48550/arXiv.1406.2572.

[2] A. Choromanska, M. Henaff, M. Mathie, G. B. Arous, and Y. LeCun, "The Loss Surfaces of Multilayer Networks," Jan. 2015, doi: arXiv.1412.0233.

[3] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul, J. M. Siskind, "Automatic differentiation in machine learning: a survey," Feb. 2018, doi: 10.48550/arXiv.1502.05767.

[4] M. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*, USA: MIT, 2017. [Online]. Available: https://doi.org/10.7551/mitpress/11301.001.0001

[5] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, Vol. 521, pp. 436 – 444. May 2015, doi: 10.1038/nature14539.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Commun. ACM*, Vol. 60, pp. 84 – 90, Jun. 2017, doi: 10.1145/3065386.

[7] Y. Shirvany, M. Hayati, and R. Moradin, "Multilayer perceptron neural networks with novel unsupervised training method for numerical solution of the partial differential equations," *Appl. Soft Comput.*, Vol. 9, pp. 20 – 29, Jan. 2009, doi: 10.1016/j.asoc.2008.02.003.

[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations," *J. Comput. Phys.*, Vol. 378, pp. 686 – 707, Feb. 2019, doi: 10.1016/j.jcp.2018.10.045.

[9] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nat. Rev. Phys.*, Vol. 3, pp. 422 – 440, Jun. 2021, doi: 10.1038/s42254-021-00314-5.

[10] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward

networks are universal approximators," *Neural Netw.*, Vol. 2, pp. 359 – 366, 1989, doi: 10.1016/0893-6080(89)90020-8.

[11] A. Pinkus, "Approximation theory of the MLP model in neural networks," *Acta Numer.*, Vol. 8, pp. 143 – 195, Jan. 1999, doi: 10.1017/S0962492900002919.

[12] A. Almqvist, "Fundamentals of Physics-Informed Neural Network Applied to Solve the Reynolds Boundary Value Problem," *Lubricants*, Vol. 9, no. 82, Aug. 2021, doi: 10.3390/lubricants9080082.

[13] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, New York, USA: JWS, 1949.

[14] T. P. Vogels and L. F. Abbott, "Signal Propagation and Logic Gating in Networks of Integrate-and-Fire Neurons," *J. Neurosci.*, Vol. 25, pp. 10786 – 10795, Nov. 2005, doi: 10.1523/JNEUROSCI.3508-05.2005.

[15] D. E. Rummelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, Vol. 323, pp. 533 – 536, 1986, doi: 10.1038/323533a0.

[16] S. -I. Amari, "Backpropagation on and stochastic gradient descent method," *Neurocomputing*, Vol. 5, pp. 185 – 196, Jun. 1993, doi: 10.1016/0925-2312(93)90006-O.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, Vol. 15, pp. 1929 – 1958, Jun. 2014.

[18] G. E. Hinton, S. Osindero, and Y.- W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, Vol. 18, pp. 1527 – 1554, 2006, doi: 10.1162/neco.2006.18.7.1527.

[19] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy Layer-Wise Training of Deep Networks," in *NIPS 2006: Int. Conf. Neural Inf. Process. Syst.*, 2006, pp. 153 – 160.

[20] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *ACM Int. Conf. Proceeding Ser.*, 2009, Vol. 382, no. 109, doi: 10.1145/1553374.1553486.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Adv. Neural Inf. Process. Syst.*, 2012, Vol. 2, pp. 1097 – 1105.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *Int. J. Comput. Vis.*, Vol. 115, pp. 211-252, Dec. 2015, doi: 10.1007/s11263-015-0816-y.

[23] M. Haenlein and A. Kaplan, "A brief history of artificial intelligence: On the past, present, and future of artificial intelligence," *Calif. Manage. Rev.*, Vol. 61, pp. 5 – 14, Aug. 2019, doi: 10.1177/0008125619864925.

[24] M. -H. Huang, R. Rust, and V. Maksimovic, "The Feeling Economy: Managing in the Next Generation of Artificial Intelligence (AI)," *Calif. Manage. Rev.*, 2019, doi: 10.1177/0008125619863436.

[25] V. Kumar, B. Rajan, R. Venkatesan, and J. Lecinski, "Understanding the role of artificial intelligence in personalized engagement marketing," *Calif. Manage. Rev.*, Vol. 61, pp. 135 – 155, Aug. 2019, doi: 10.1177/0008125619859317.

[26] J. K. -U. Brock and F. v. Wangenheim, "Demystifying Ai: What digital transformation leaders can teach you about realistic artificial intelligence," *Calif. Manage. Rev.*, Vol. 61, pp. 110 – 134, Aug. 2019, doi: 10.1177/1536504219865226.

[27] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Trans. Neural Netw.*, Vol. 9, pp. 987 – 1000, 1998, doi: 10.1109/72.712178.

[28] P. Sharma, W. T. Chung, B. Akoush, and M. Ihme, "A Review of Physics-Informed Machine Learning in Fluid Mechanics," *Energies*, Vol. 16, no. 2343, Mar. 2023, doi: 10.3390/en16052343.

[29] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, Vol. 378, pp. 686 – 707, Feb. 2019, doi: 10.1016/j.jcp.2018.10.045.

[30] C. Bajaj, L. McLennan, T. Andeen, and A. Roy, "Recipes for when physics fails: recovering robust learning of physics informed neural networks," *Mach. Learn.: Sci. Technol.*, Vol. 4, no. 015013, Mar. 2023, doi: 10.1088/2632-2153/acb416.

[31] M. Rasht-Behesht, C. Huber, K. Shukla, and G. E. Karniadakis, "Physics-Informed Neural Networks (PINNs) for Wave Propagation and Full Waveform Inversions," *J. Geophys. Res.*, Vol. 127, no. e2021JB023120, May 2022, doi: 10.1029/2021JB023120.

[32] S. Alkhadhr and M. Almekkawy, "Wave Equation Modeling via Physics-Informed Neural Networks: Models of Soft and Hard Constraints for Initial and Boundary Conditions," *J. Sensors*, Vol. 23, no. 2792, Mar. 2023, doi: 10.3390/s23052792.

[33] N. Zobeiry and K. D. Humfeld, "A physics-informed machine learning approach for solving heat transfer equation in advanced manufacturing and engineering applications," *Eng. Appl. Artif. Intell.*, Vol. 101, no. 104232, May 2021, doi: 10.1016/j.engappai.2021.104232.

[34] F. Heldmann, S. Berkhahn, M. Ehrhardt, K. Klamroth, "PINN training using biobjective optimization: The trade-off between data loss and residual loss," *J. Comput. Phys.*, Vol. 488, no. 112211, Sept. 2023, doi: 10.1016/j.jcp.2023.112211.

[35] M. Usama, R. Ma, J. Hart, and M. Wojcik, "Physics-Informed Neural Networks (PINNs)-Based Traffic State Estimation: An Application to Traffic Networks," *Algorithms*, Vol. 15, no. 447, Dec. 2022, doi: 10.3390/a15120447.

[36] N. V. Jagtap, M. K. Mudunuru, and K. B. Nakshatrala, "CoolPINNs: A physics-informed neural network modeling of active cooling in vascular systems," *Appl. Math. Model*, Vol. 122, pp. 265 – 287, Oct. 2023, doi: 10.1016/j.apm.2023.04.020.

[37] Y. Zhao, L. Guo, and P. P. L. Wong, "Application of physics-informed neural network in the analysis of hydrodynamic lubrication," *Friction*, Vol. 11, pp. 1253 – 1264, Jul. 2023, doi: 10.1007/s40544-022-0658-x.

[38] I. Dincer and S. Osamah, “Heat Transfer Aspects of Energy”, in Comprehensive Energy Systems, vol. 1-5, Elsevier Inc., 2018, sec. 1.10, pp. 422 – 477. [Online]. Available: https://doi.org/10.1016/B978-0-12-809597-3.00109-7

[39] H. J. Ricardo, “The numerical approximation of solutions”, in A Modern Introduction to Differential Equations, 3rd ed., USA: Academic, 2020, ch. 3, pp. 111 - 142. [Online]. Available: https://doi.org/10.1016/B978-0-12-818217-8.00010-5

[40] K. A. Hoffman and S. T. Chiang, “Parabolic Partial Differential Equations”, in Computational Fluid Dynamics, 4th ed., Wichita, Kansas, USA: Eng. Educ. Syst., 2000, ch. 3, pp. 60 - 112.

LAMPIRAN

1. Kode program ANN untuk kasus satu dimensi

```
%{
A MATLAB program to calculate 1D Heat Transfer problem. Neural
Network
is assigned to solve the Partial Differential Equations in problem.
1D Heat Transfer problem is described as "dy/dt = alpha*(d2y/dx2)".

Created by:
Theodoret Putra Agatho
University of Atma Jaya Yogyakarta
Department of Informatics
09/05/2023

Disclaimer:
This program writing is influenced heavily by following code:
Andreas Almqvist (2023). Physics-informed neural network solution of
2nd order ODE:s
(https://www.mathworks.com/matlabcentral/fileexchange/96852-physics-
informed-neural-network-solution-of-2nd-order-ode-s),
MATLAB Central File Exchange. Retrieved May 11, 2023.
%}

clear all; clc; close all;
%% Initialization

% Hyperparameters
alpha = 1.0;

% Time
dt = 0.01;
time = 0.1; % time max
epoch_time = floor(time/dt);
time_step = 2; % graph changing every mod(i,time_step)==0

% Grid 1D
Nx = 41; % N-many elements in x vector
xmin = 0; xmax = 1; % boundary x vector
x = linspace(xmin,xmax,Nx); % Grid
dx = x(2) - x(1);

% Neural Network
learning_rate = 0.001;
epoch_learning = 1000;
batch = 100;
Nn = 40; % N-many neuron
% weightes and biases
min = -2; max = 2; % boundary weights and biases
w0 = min + rand(Nn,1)*(abs(min)+abs(max));
b0 = min + rand(Nn,1)*(abs(min)+abs(max));
w1 = min + rand(Nn,1)*(abs(min)+abs(max));
```

```
47. b1 = min + rand(1)*(abs(min)+abs(max));
48. params = [w0;b0;w1;b1];
49. % initial_params = params;
50.
51. % Initial values
52. U = sin(pi*x); % consider "U" as current y and "y" as next y
53.
54.
55.
56. %% Other solutions
57. % Numerical solution - Finite Difference Method
58. Ufd = U;
59. s = alpha*dt/dx^2;
60.
61. A = eye([Nx,Nx]);
62. for i = 2:Nx-1
63.     for j = 2:Nx-1
64.         if A(i,j) == 1
65.             A(i,j) = 1 + 2*s;
66.             A(i,j-1) = -s;
67.             A(i,j+1) = -s;
68.         end
69.     end
70. end
71.
72. t0fd = tic; % stopwatch start
73. for i = 1:epoch_time
74.     Ufd = Ufd/A;
75.     Ufd(1) = 0; Ufd(end) = 0; % boundary conditions
76. end
77. tfd = toc(t0fd); % stopwatch end
78.
79. % Exact solution
80. Ue = sin(pi.*x).*exp(1).^(-pi^2.*time);
81.
82.
83.
84. %% Neural Network training
85. t0nn = tic; % stopwatch start
86. for i = 1:epoch_time
87.     disp(i)
88.     for j = 1:epoch_learning
89.         params =
    training(params,U,x,Nn,alpha,dt,batch,learning_rate);
90.     end
91.     [y,~,~] = predict(params,x,Nn);
92.     U = y;
93.     if mod(i,time_step)==0
94.         figure(gcf);plot(x,U);ylim([0 1]);
95.     end
96. end
97. tnn = toc(t0nn); % stopwatch end
98.
99.
```

```
100.
101.   %% Results and Visualisations
102.   % Visualisations
103.   figure(2); clf;
104.   plot(x,U,'r--o'); % Neural Network prediction
105.   hold on;
106.   plot(x,Ufd,'g--'); % Finite Difference solution
107.   plot(x,Ue,'k'); % Exact solution
108.   lh = legend('PINN prediction','Finite Difference solution','Exact
   solution');
109.   set(lh,'interpreter','latex','fontsize',16,'location','northwest'
   );
110.   set(gca,'fontsize',16)
111.   hold off;
112.
113.   % Error
114.   error_nn = mean((U-Ue).^2);
115.   error_fd = mean((Ufd-Ue).^2);
116.
117.
118.
119.   %% Functions (Neural Network)
120.   function y = mySigmoid(x)
121.       y = 1./(1 + exp(-x));
122.   end
123.
124.   function params =
   training(params,U,x,Nn,alpha,dt,batch,learning_rate)
125.       w0 = params(1:Nn);
126.       b0 = params(Nn+1:Nn*2);
127.       w1 = params(Nn*2+1:end-1);
128.       b1 = params(end);
129.
130.       for j = 1:batch
131.           % Pick a random data point for current batch
132.           i = randi(length(x));
133.           xi = x(i);
134.           Ui = U(i);
135.
136.           % Sigmoid function derivatives with w0, b0, and xi inputs
137.           s = mySigmoid(w0*xi + b0);
138.           dsdx = s.*(1 - s);
139.           d2sdx2 = dsdx.*(1 - 2*s);
140.           d3sdx3 = d2sdx2.*(1 - 2*s) - 2*dsdx.^2;
141.           % at boundary
142.           s0 = mySigmoid(w0*x(1) + b0);
143.           s1 = mySigmoid(w0*x(end) + b0);
144.
145.           % Prediction current batch
146.           y = sum(w1.*s) + b1;
147.           % dydx = sum(w0.*w1.*dsdx); % unused
148.           d2ydx2 = sum(w0.^2.*w1.*d2sdx2);
149.           % at boundary
150.           y0 = sum(w1.*s0) + b1;
```

```
151.            y1 = sum(w1.*s1) + b1;
152.
153.            % y derivatives to weights and biases
154.            dydw0 = w1.*dsdx.*xi;
155.            dydb0 = w1.*dsdx;
156.            dydw1 = s;
157.            dydb1 = 1;
158.
159.            % dydx derivatives to weights and biases - unused
160.            % dypdw0 = w1.*dsdx + w0.*w1.*d2sdx2.*xi; % yp = dydx
161.            % dypdb0 = w0.*w1.*d2sdx2;
162.            % dypdw1 = w0.*dsdx;
163.            % dypdb1 = 0;
164.
165.            % d2ydx2 derivatives to weights and biases
166.            dyppdw0 = 2*w0.*w1.*d2sdx2 + w0.^2.*w1.*d3sdx3.*xi; % ypp
    = d2ydx2
167.            dyppdb0 = w0.^2.*w1.*d3sdx3;
168.            dyppdw1 = w0.^2.*d2sdx2;
169.            dyppdb1 = 0;
170.
171.            ds0dx = s0.*(1 - s0);
172.            % y0 derivatives to weights and biases
173.            dy0dw0 = w1.*ds0dx.*x(1);
174.            dy0db0 = w1.*ds0dx;
175.            dy0dw1 = s0;
176.            dy0db1 = 1;
177.
178.            ds1dx = s1.*(1 - s1);
179.            % y1 derivatives to weights and biases
180.            dy1dw0 = w1.*ds1dx.*x(end);
181.            dy1db0 = w1.*ds1dx;
182.            dy1dw1 = s1;
183.            dy1db1 = 1;
184.
185.            % Weights and biases update
186.            % l = mean((y - dt*alpha*d2ydx2 - Ui).^2) + (y0)^2 +
    (y1)^2; % loss function
187.            w0 = w0 - learning_rate*(2*(y - dt*alpha*d2ydx2 - Ui)*
    ...
188.                (dydw0 - dt*alpha*dyppdw0) + 2*y0*dy0dw0 +
    2*y1*dy1dw0);
189.            b0 = b0 - learning_rate*(2*(y - dt*alpha*d2ydx2 - Ui)*
    ...
190.                (dydb0 - dt*alpha*dyppdb0) + 2*y0*dy0db0 +
    2*y1*dy1db0);
191.            w1 = w1 - learning_rate*(2*(y - dt*alpha*d2ydx2 - Ui)*
    ...
192.                (dydw1 - dt*alpha*dyppdw1) + 2*y0*dy0dw1 +
    2*y1*dy1dw1);
193.            b1 = b1 - learning_rate*(2*(y - dt*alpha*d2ydx2 - Ui)*
    ...
194.                (dydb1 - dt*alpha*dyppdb1) + 2*y0*dy0db1 +
    2*y1*dy1db1);
```

```
195.        end
196.        params = [w0;b0;w1;b1];
197.    end
198.
199.    function [y,dydx,d2ydx2] = predict(params,x,Nn)
200.        w0 = params(1:Nn);
201.        b0 = params(Nn+1:Nn*2);
202.        w1 = params(Nn*2+1:end-1);
203.        b1 = params(end);
204.
205.        Nx = length(x);
206.        w0 = repmat(w0,1,Nx);
207.        b0 = repmat(b0,1,Nx);
208.        w1 = repmat(w1,1,Nx);
209.        x = repmat(x,Nn,1);
210.
211.        % Sigmoid function derivative with w0, b0, and x inputs
212.        s = mySigmoid(w0.*x + b0);
213.        dsdx = s.*(1 - s);
214.        d2sdx2 = dsdx.*(1 - 2*s);
215.
216.        % Prediction
217.        y = sum(w1.*s) + b1;
218.        dydx = sum(w0.*w1.*dsdx);
219.        d2ydx2 = sum(w0.^2.*w1.*d2sdx2);
220.    end
221.
```

2. Kode program ANN untuk kasus dua dimensi

```
%{
A MATLAB program to calculate 2D Heat Transfer problem. Neural
Network
is assigned to solve the Partial Differential Equations in problem.
2D Heat Transfer problem is described as "du/dt = alpha*(d2u/dx2 +
d2u/dy2)".

Created by:
Theodoret Putra Agatho
University of Atma Jaya Yogyakarta
Department of Informatics
27/07/2023

Disclaimer:
This program writing is influenced heavily by following code:
Andreas Almqvist (2023). Physics-informed neural network solution of
2nd order ODE:s
(https://www.mathworks.com/matlabcentral/fileexchange/96852-physics-
informed-neural-network-solution-of-2nd-order-ode-s),
MATLAB Central File Exchange. Retrieved May 11, 2023.
%}

clear all; clc; close all;
%% Initialization

% Hyperparameters
alpha = 0.1;

% Time
dt = 0.1;
time = 1.0; % time max
epoch_time = floor(time/dt);
time_step = 2; % graph changing every mod(i,time_step)==0

% Grid 2D
Nx = 21; % N-many elements in x vector
Ny = 11; % N-many elements in x vector
% x-vector
xmin = 0; xmax = 4; % boundary x vector
xline = linspace(xmin,xmax,Nx); % Grid
x = zeros(Ny,Nx);
for i = 1:Ny
    x(i,:) = xline;
end
dx = xline(2) - xline(1);

% y-vector
ymin = 0; ymax = 2; % boundary x vector
yline = linspace(ymin,ymax,Ny); % Grid
y = zeros(Ny,Nx);
for i = 1:Nx
    y(:,i) = yline;
end
```

```
dy = yline(2) - yline(1);

% Neural Network
learning_rate = 0.001;
epoch_learning = 512;
batch = 128;
Nn = 32; % N-many neuron
% weightes and biases
min = -2; max = 2; % boundary weights and biases
w0x = min + rand(Nn,1)*(abs(min)+abs(max));
w0y = min + rand(Nn,1)*(abs(min)+abs(max));
b0 = min + rand(Nn,1)*(abs(min)+abs(max));
w1 = (min + rand(Nn,1)*(abs(min)+abs(max)));
b1 = (min + rand(1)*(abs(min)+abs(max)));
params = [w0x;w0y;b0;w1;b1];
% initial_params = params;

% Initial values
U = 100.*ones(Ny,Nx)./300; % consider "U" as current y and "y" as
   next y
U(1,:) = 300.*ones(1,Nx)./300;
U(end,:) = 300.*ones(1,Nx)./300;


%% Other solutions
% Analytical (/exact) Solution
L = ymax;
Uetemp = zeros(Ny,Nx);

mmax = 201;
for m = 1:mmax
    Uetemp = Uetemp + exp(-(m*pi/L).^2.*alpha.*time).*((1-(-
   1).^m)./(m.*pi)).*sin(m*pi*y/L);
end
Ue = (300 + 2*(100-300).*Uetemp)./300;

% Numerical solution - Finite Difference Method
Ufd = U(2:Ny-1,2:Nx-1);
sx = alpha*dt/dx^2;
sy = alpha*dt/dy^2;

Ni = (Ny-2)*(Nx-2);
A = eye(Ni,Ni);
for i = 1:Ni
    for j = 1:Ni
        if A(i,j) == 1
            A(i,j) = 1 + 2*sx + 2*sy;
            if j > 1 && mod(j,(Ny-2)) ~= 1
                A(i,j-1) = -sx;
            end
            if j < Ni && mod(j,(Ny-2)) ~= 0
                A(i,j+1) = -sx;
            end
            if j > (Ny-2)
```

```
                    A(i,j-(Ny-2)) = -sy;
                end
                if j <= Ni-(Ny-2)
                    A(i,j+(Ny-2)) = -sy;
                end
            end
        end
    end

    % Boundary
    By = zeros(Ny-2,Nx-2);
    Bx = zeros(Ny-2,Nx-2);
    Bx(end,:) = 300.*ones(1,Nx-2)./300;
    Bx(1,:) = 300.*ones(1,Nx-2)./300;
    By(1,:) = 300.*ones(1,Nx-2)./300;
    By(end,:) = 300.*ones(1,Nx-2)./300;
    By = reshape(By,[],Ni);
    Bx = reshape(Bx,[],Ni);
    Ufd = reshape(Ufd,[],Ni);

    t0fd = tic; % stopwatch start
    for i = 1:epoch_time
        Bx = Ufd;
        Bx = reshape(Bx,[Ny-2,Nx-2]);
        Bx(:,2:end-1) = zeros(Ny-2,Nx-4);
        Bx = reshape(Bx,[],Ni);
        Ufd = (Ufd + Bx*sx + By*sy)/A;
    end
    tfd = toc(t0fd); % stopwatch end

    Ufd = reshape(Ufd,[Ny-2,Nx-2]);
    Utemp = 100.*ones(Ny,Nx)./300;
    Utemp(2:Ny-1,2:Nx-1) = Ufd;
    Utemp(end,:) = 300.*ones(1,Nx)./300;
    Utemp(1,:) = 300.*ones(1,Nx)./300;
    Utemp(:,1) = Utemp(:,2);
    Utemp(:,end) = Utemp(:,end-1);
    Ufd = Utemp;



    %% Neural Network training
    t0nn = tic; % stopwatch start
    for i = 1:epoch_time
        disp(i)
        for j = 1:epoch_learning
            params =
training(params,U,x,y,Nn,alpha,dt,batch,learning_rate);
        end
        [u,~,~] = predict(params,x,y,Nn);
        U = u;
        if mod(i,time_step)==0
            figure(gcf);surf(x,y,U);%ylim([0 1]);
        end
```

```
155.    end
156.    tnn = toc(t0nn); % stopwatch end
157.    % U = U.*300;
158.    surf(x,y,U);title('Neural Network');
159.
160.
161.    %% Results and Visualisations
162.    % Visualisations
163.    figure(2);surf(x,y,Ufd);title('Finite Difference');
164.
165.    % Error
166.    error_nn = mean((U-Ue).^2,'all');
167.    error_fd = mean((Ufd-Ue).^2,'all');
168.    figure(3);surf(x,y,U-Ue);title('Neural Network Error');
169.    figure(4);surf(x,y,Ufd-Ue);title('Finite Difference Error');
170.
171.    % mean((U(end,:)-1).^2+(U(1,:)-1).^2+(U(:,1)-
   U(:,2)).^2+(U(:,end)-U(:,end-1)).^2,'all')
172.
173.
174.
175.    %% Functions (Neural Network)
176.    function y = mySigmoid(x)
177.        y = 1./(1 + exp(-x));
178.    end
179.
180.    function params =
   training(params,U,x,y,Nn,alpha,dt,batch,learning_rate)
181.        w0x = params(1:Nn);
182.        w0y = params(Nn+1:Nn*2);
183.        b0 = params(Nn*2+1:Nn*3);
184.        w1 = params(Nn*3+1:end-1);
185.        b1 = params(end);
186.
187.        Ny = size(y,1);
188.        Nx = size(x,2);
189.        btop = 300.*ones(1,Nx)./300;
190.        bbottom = 300.*ones(1,Nx)./300;
191.
192.        % Boundary weights and biases
193.        w0xbx = repmat(w0x,1,Ny); w0ybx = repmat(w0y,1,Ny);
194.        w0xby = repmat(w0x,1,Nx); w0yby = repmat(w0y,1,Nx);
195.
196.        xleft = repmat(x(:,1).',Nn,1) ; xright  =
   repmat(x(:,end).',Nn,1);
197.        xnleft = repmat(x(:,2).',Nn,1) ; xnright  = repmat(x(:,end-
   1).',Nn,1);
198.        xtop  = repmat(x(1,:),Nn,1)   ; xbottom =
   repmat(x(end,:),Nn,1);
199.
200.        yleft = repmat(y(:,1).',Nn,1) ; yright  =
   repmat(y(:,end).',Nn,1);
201.        ynleft = repmat(y(:,2).',Nn,1) ; ynright  = repmat(y(:,end-
   1).',Nn,1);
```

```
202.       ytop  = repmat(y(1,:),Nn,1)   ; ybottom =
   repmat(y(end,:),Nn,1);
203.       for k = 1:batch
204.   %         disp(k)
205.           % Pick a random data point for current batch
206.           j = randi(Nx);
207.           i = randi(Ny);
208.           xi = x(i,j);
209.           yi = y(i,j);
210.           Ui = U(i,j);
211.
212.           % Sigmoid function derivatives with w0, b0, and xi inputs
213.           s = mySigmoid(w0x*xi + w0y*yi + b0);
214.           sp = s.*(1 - s);
215.           spp = sp.*(1 - 2*s);
216.           sppp = spp.*(1 - 2*s) - 2*sp.^2;
217.
218.           % at boundary
219.           sleft = mySigmoid(w0xbx.*xleft + w0ybx.*yleft + b0); %
   [Nn,Ny]
220.           sright = mySigmoid(w0xbx.*xright + w0ybx.*yright + b0);
221.           stop = mySigmoid(w0xby.*xtop + w0yby.*ytop + b0); %
   [Nn,Nx]
222.           sbottom = mySigmoid(w0xby.*xbottom + w0yby.*ybottom +
   b0);
223.
224.           snleft = mySigmoid(w0xbx.*xnleft + w0ybx.*ynleft + b0); %
   [Nn,Ny]
225.           snright = mySigmoid(w0xbx.*xnright + w0ybx.*ynright +
   b0);
226.
227.           % Prediction current batch
228.           v = sum(w1.*s) + b1;
229.           % dvdx = sum(w0x.*w1.*sp); % unused
230.           d2vdx2 = sum(w0x.^2.*w1.*spp);
231.           % dvdy = sum(w0y.*w1.*sp); % unused
232.           d2vdy2 = sum(w0y.^2.*w1.*spp);
233.           % at boundary
234.           vleft = sum(w1.*sleft) + b1; % [1,Ny]
235.           vright = sum(w1.*sright) + b1;
236.           vtop = sum(w1.*stop) + b1; % [1,Nx]
237.           vbottom = sum(w1.*sbottom) + b1;
238.
239.           bleft = sum(w1.*snleft) + b1; % [1,Ny]
240.           bright = sum(w1.*snright) + b1;
241.
242.
243.           % y derivatives to weights and biases
244.           dvdw0x = w1.*sp.*xi;
245.           dvdw0y = w1.*sp.*yi;
246.           dvdb0 = w1.*sp;
247.           dvdw1 = s;
248.           dvdb1 = 1;
249.
```

```
250.            % dvdx derivatives to weights and biases - unused
251.            % dvpdw0x = w1.*sp + w0x.*w1.*spp.*xi;
252.            % dvpdw0y = w0x.*w1.*spp.*yi;
253.            % dvpdb0 = w0x.*w1.*spp;
254.            % dvpdw1 = w0x.*sp;
255.            % dvpdb1 = 0;
256.
257.            % d2vdx2 derivatives to weights and biases
258.            dvxppdw0x = 2*w0x.*w1.*spp + w0x.^2.*w1.*sppp.*xi; % ypp
    = d2ydx2
259.            dvxppdw0y = w0x.^2.*w1.*sppp.*yi;
260.            dvxppdb0 = w0x.^2.*w1.*sppp;
261.            dvxppdw1 = w0x.^2.*spp;
262.            dvxppdb1 = 0;
263.
264.            % d2vdy2 derivatives to weights and biases
265.            dvyppdw0x = w0y.^2.*w1.*sppp.*xi; % ypp = d2ydx2
266.            dvyppdw0y = 2*w0y.*w1.*spp + w0y.^2.*w1.*sppp.*yi;
267.            dvyppdb0 = w0y.^2.*w1.*sppp;
268.            dvyppdw1 = w0y.^2.*spp;
269.            dvyppdb1 = 0;
270.
271.            dsldx = sleft.*(1 - sleft); % [Nn,Ny]
272.            % vleft derivatives to weights and biases
273.            dvldw0x = dsldx.*w1.*xleft; % [Nn,Ny]
274.            dvldw0y = dsldx.*w1.*yleft;
275.            dvldb0 = w1.*dsldx;
276.            dvldw1 = sleft;
277.            dvldb1 = 1;
278.
279.            dsrdx = sright.*(1 - sright);
280.            % vright derivatives to weights and biases
281.            dvrdw0x = dsrdx.*w1.*xright;
282.            dvrdw0y = dsrdx.*w1.*yright;
283.            dvrdb0 = w1.*dsrdx;
284.            dvrdw1 = sright;
285.            dvrdb1 = 1;
286.
287.            dstdx = stop.*(1 - stop); % [Nn,Nx]
288.            % vleft derivatives to weights and biases
289.            dvtdw0x = dstdx.*w1.*xtop; % [Nn,Nx]
290.            dvtdw0y = dstdx.*w1.*ytop;
291.            dvtdb0 = w1.*dstdx;
292.            dvtdw1 = stop;
293.            dvtdb1 = 1;
294.
295.            dsbdx = sbottom.*(1 - sbottom);
296.            % vleft derivatives to weights and biases
297.            dvbdw0x = dsbdx.*w1.*xbottom;
298.            dvbdw0y = dsbdx.*w1.*ybottom;
299.            dvbdb0 = w1.*dsbdx;
300.            dvbdw1 = sbottom;
301.            dvbdb1 = 1;
302.
```

```
303.
304.           dslndx = snleft.*(1 - snleft); % [Nn,Ny]
305.           % vleft derivatives to weights and biases
306.           dvlndw0x = dslndx.*w1.*xnleft; % [Nn,Ny]
307.           dvlndw0y = dslndx.*w1.*ynleft;
308.           dvlndb0 = w1.*dslndx;
309.           dvlndw1 = snleft;
310.           dvlndb1 = 1;
311.
312.           dsrndx = snright.*(1 - snright);
313.           % vright derivatives to weights and biases
314.           dvrndw0x = dsrndx.*w1.*xnright;
315.           dvrndw0y = dsrndx.*w1.*ynright;
316.           dvrndb0 = w1.*dsrndx;
317.           dvrndw1 = snright;
318.           dvrndb1 = 1;
319.
320.
321.           % Weights and biases update
322.           % l = mean((y - dt*alpha*d2udx2*d2udy2 - Ui).^2) + (y0)^2
   + (y1)^2; % loss function
323.           b1 = b1 - learning_rate*(2*(v - dt*alpha*(d2vdx2+d2vdy2)
   - Ui)* ...
324.                (dvdb1 - dt*alpha*(dvxppdb1+dvyppdb1)) +
   sum(2*(vleft-bleft).*(dvldb1-dvlndb1)) + ...
325.                sum(2*(vright-bright).*(dvrdb1-dvrndb1)) +
   sum(2*(vtop-btop).*dvtdb1) + sum(2*(vbottom-bbottom).*dvbdb1));
326.           vleft = repmat(vleft,Nn,1); vright = repmat(vright,Nn,1);
327.           vtop = repmat(vtop,Nn,1); vbottom = repmat(vbottom,Nn,1);
328.           w0x = w0x - learning_rate*(2*(v -
   dt*alpha*(d2vdx2+d2vdy2) - Ui)* ...
329.                (dvdw0x - dt*alpha*(dvxppdw0x+dvyppdw0x)) +
   sum(2*(vleft-bleft).*(dvldw0x-dvlndw0x),2) + ...
330.                sum(2*(vright-bright).*(dvrdw0x-dvrndw0x),2) +
   sum(2*(vtop-btop).*dvtdw0x,2) + sum(2*(vbottom-
   bbottom).*dvbdw0x,2));
331.           w0y = w0y - learning_rate*(2*(v -
   dt*alpha*(d2vdx2+d2vdy2) - Ui)* ...
332.                (dvdw0y - dt*alpha*(dvxppdw0y+dvyppdw0y)) +
   sum(2*(vleft-bleft).*(dvldw0y-dvlndw0y),2) + ...
333.                sum(2*(vright-bright).*(dvrdw0y-dvrndw0y),2) +
   sum(2*(vtop-btop).*dvtdw0y,2) + sum(2*(vbottom-
   bbottom).*dvbdw0y,2));
334.           b0 = b0 - learning_rate*(2*(v - dt*alpha*(d2vdx2+d2vdy2)
   - Ui)* ...
335.                (dvdb0 - dt*alpha*(dvxppdb0+dvyppdb0)) +
   sum(2*(vleft-bleft).*(dvldb0-dvlndb0),2) + ...
336.                sum(2*(vright-bright).*(dvrdb0-dvrndb0),2) +
   sum(2*(vtop-btop).*dvtdb0,2) + sum(2*(vbottom-bbottom).*dvbdb0,2));
337.           w1 = w1 - learning_rate*(2*(v - dt*alpha*(d2vdx2+d2vdy2)
   - Ui)* ...
338.                (dvdw1 - dt*alpha*(dvxppdw1+dvyppdw1)) +
   sum(2*(vleft-bleft).*(dvldw1-dvlndw1),2) + ...
339.                sum(2*(vright-bright).*(dvrdw1-dvrndw1),2) +
```

```
   sum(2*(vtop-btop).*dvtdw1,2) + sum(2*(vbottom-bbottom).*dvbdw1,2));
340.        end
341.        params = [w0x;w0y;b0;w1;b1];
342.    end
343.
344.    function [y,dydx,d2ydx2] = predict(params,x,y,Nn)
345.        w0x = params(1:Nn);
346.        w0y = params(Nn+1:Nn*2);
347.        b0 = params(Nn*2+1:Nn*3);
348.        w1 = params(Nn*3+1:end-1);
349.        b1 = params(end);
350.
351.        Nx = size(x,2);
352.        Ny = size(y,1);
353.        Ni = Nx*Ny;
354.        w0x = repmat(w0x,1,Ni);
355.        w0y = repmat(w0y,1,Ni);
356.        b0 = repmat(b0,1,Ni);
357.        w1 = repmat(w1,1,Ni);
358.        x = repmat(reshape(x,[1,Ni]),Nn,1);
359.        y = repmat(reshape(y,[1,Ni]),Nn,1);
360.
361.        % Sigmoid function derivative with w0, b0, and x inputs
362.        s = mySigmoid(w0x.*x + w0y.*y + b0);
363.        dsdx = s.*(1 - s);
364.        d2sdx2 = dsdx.*(1 - 2*s);
365.
366.        % Prediction
367.        y = sum(w1.*s) + b1;
368.        dydx = sum(w0x.*w1.*dsdx);
369.        d2ydx2 = sum(w0x.^2.*w1.*d2sdx2);
370.        y = reshape(y,[Ny,Nx]);
371.    end
372.
```

3. Naskah konferensi InCASST 2023

Artificial Neural Network in 1D Heat Transfer Problem

Theodoret P. Agatho[1, a)] and Pranowo[1, b)]

[1]*Department of Informatics, Faculty of Industrial Technology, University of Atma Jaya Yogyakarta, Babarsari St. No. 43, Yogyakarta, 55281, Indonesia*

[a)] Corresponding author: 190710233@students.uajy.ac.id
[b)]pranowo.uajy.ac.id

Abstract. In this paper, Artificial Neural Network (ANN) was used to help solve one-dimension unsteady conduction heat transfer problem. In this case, the derivatives of space were discretized by ANN and the time was integrated by Euler's Method. All calculation programs were written in Matlab. Subsequently, the parameter and runtime of ANN were observed and the results were compared to Finite Difference Method (FDM) results and exact solution. A number of numerical experiments has been performed and ANN showed high degree of accuracy (mean square error 10^{-4} and below) and capability to outperform the accuracy from FDM in certain cases.

INTRODUCTION

ANN is well known for its capability to solve complex problems by taking the patterns in data where rigorous rules can't be defined, which means it can't be ruled rigidly. It is made possible through "learning" or "training", where the weight and bias adjustment of ANN is done iteratively based on error between prediction and observation data. Hence, this predominance behavior is seen as promising since it means many problems from variety fields can be solved by ANN alone as long as sufficient observed data is available. However, further implementation may lead to bigger data requirement and be susceptible to overfitting over observational data. The limitation on conventional ANN capability to process raw data [1] encourages last decade research further to enhance ANN performability.

Recent scientific research in mathematical modeling found that this problem can be mitigated by taking physic law into consideration. In this case, by satisfying the governing equation described by Partial Differential Equations (PDEs) rather than solely relying on data. This method makes ANN comparable to numerical discretization, such as FDM, which is used to solve PDEs. Nevertheless, solving PDEs with ANN has advantages over numerical approach, such as (1) its capability to acquire result when the numerical approaches are failing because the law is partially known [2] or the introduction of noise [3] and (2) its potential to yield more accurate result [4].

The applications of physic law to ANN, known as part of Physic Informed Neural Network (PINN) family, have been studied extensively in following refs [5-13]. ANN has been used to solve FDM in wave equations [5,6], heat transfer equations [7], and Navier-Stokes equations [8,9]. In addition, it's able to solve more complicated mathematical cases such as fractional PDE [10] and high-dimensional PDE [11].

From preceding discussion, it's clear that this matter is well-established and further research is still work on. This paper tries to take a closer look at the capability and limitation of ANN in 1D Heat Transfer Problem while presenting the mathematical application of it.

The paper was divided into 4 sections: background and ideas were introduced in section 1. Section 2 covered basic theory and formula of ANN and time-dependent PDEs. Results were discussed later in section 3 and conclusions were drawn in section 4.

THEORY BACKGROUND

Heat Transfer

Consider the following simple one-dimension heat transfer equation, where the thermal conductivity, density, and heat capacity are constant over time:

$$\frac{\partial y}{\partial t} = \alpha \frac{\partial^2 y}{\partial x^2}, \quad 0 < x < 1, y(0) = 0 = y(1), \tag{1}$$

Where $\alpha = 1$ and the initial value are $y_t = \sin(\pi x)$. Thus, the exact solution is $y_t = \sin(\pi x) e^{-(\pi^2 t)}$. When time is integrated by applying implicit Euler's method, equation (1) turns as follow:

$$\frac{y_{n+1} - y_n}{\Delta t} = \frac{\partial y}{\partial t}, \tag{2}$$

$$\frac{y_n}{\Delta t} = \frac{y_{n+1}}{\Delta t} - \alpha \frac{\partial^2 y_{n+1}}{\partial x^2}, \tag{3}$$

The slight value of Δt may cause overflow in computer calculations; therefore, preceding equation is modified as follows:

$$y_n = y_{n+1} - \Delta t \left(\alpha \frac{\partial^2 y_{n+1}}{\partial x^2} \right) \tag{4}$$

Artificial Neural Network

To solve the problem, a feed forward neural network was used. The architecture of this ANN was consisted of one input node x, one hidden layer consists of N-many nodes, and one output node y. The Sigmoid function was used as activation function (φ) in hidden layer and gradient decent as the optimizer. It's worth to take note that the differentiation of network output y must be able to satisfy the underlining physic law. Following is the used ANN architecture.

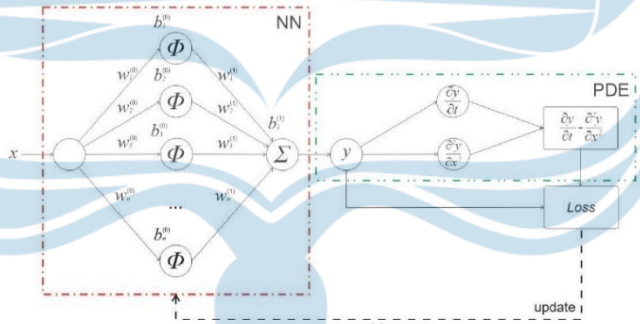

FIGURE 1. ANN Architecture

From preceding equation (4), ANN was used to predict y_{n+1}, where y_n is known constant, substituting FD. Therefore y_{n+1} was calculated with following formulation:

$$y_{n+1} = w^{(1)}\varphi\left(w^{(0)}x + b^{(0)}\right) + b^{(1)} \tag{5}$$

Where weight and bias values were initialized randomly in interval [-2,2]. Weight and value adjustment are carried out resemble to ordinary ANN, which based on loss, but preceding physic law, which contains boundary conditions, was also taken into consideration in loss, thus the loss function is as follow:

$$l = \left\langle \left(y_{n+1} - \Delta t\left(\alpha \frac{\partial^2 y_{n+1}}{\partial x^2} \right) - y_n \right)^2 \right\rangle + \left(y_{n+1}(0)\right)^2 + \left(y_{n+1}(1)\right)^2, \tag{6}$$

Consequently, the gradient descent, which in this case is for the weight, is given as:

$$\frac{\partial l}{\partial w} = \left\langle 2\left(y_{n+1} - \Delta t\left(\alpha\nabla^2 y_{n+1}\right) - y_n\right)\left(\frac{\partial y_{n+1}}{\partial w} - \Delta t\left(\alpha \frac{\partial \nabla^2 y_{n+1}}{\partial w} \right) \right) \right\rangle + 2\left(y_{n+1}(0)\right)\left(\frac{\partial y_{n+1}(0)}{\partial w} \right) + 2\left(y_{n+1}(1)\right)\left(\frac{\partial y_{n+1}(1)}{\partial w} \right), \tag{7}$$

Where,

$$\nabla^2 y_{n+1} = \frac{\partial^2 y_{n+1}}{\partial x^2} = w^{(1)}\left(w^{(0)}\right)^2 \varphi\left(w^{(0)}x + b^{(0)}\right), \tag{8}$$

$$\nabla^2 \varphi\left(w^{(0)}x + b^{(0)}\right) = \left(1 - \varphi\left(w^{(0)}x + b^{(0)}\right)\right)\left(1 - 2\varphi\left(w^{(0)}x + b^{(0)}\right)\right), \tag{9}$$

And x is coordinate input. Equation (7) is also true toward bias (b) where weight (w) is substituted. The next prediction over the change of time, y_{n+1} before become current y_n and the learning process is repeated. This process is repeated until accumulated time reached the targeted time. Subsequent flowchart shows the program flow.

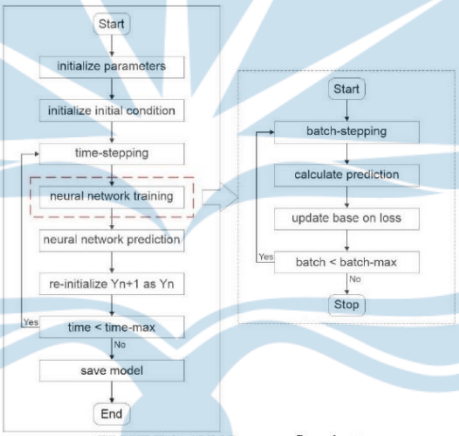

FIGURE 2. ANN Program flowchart

RESULT AND DISCUSSION

The calculations were done by computer using MATLAB program. ANN managed to achieve accurate result with low error compared to numerical result error toward exact solution. Various parameters tunings have been tested and its average of runtime and Mean Square Error (MSE) was observed.

TABLE 1. ANN and FD results at total time is 0.1 and the total node of x is 41, while the learning rate of ANN is 0.001.

dt	Total neural	Total epoch	Total batch	Neural Network		Finite Difference	
				Average runtime (sec)	Average MSE	Average runtime (sec)	Average MSE
0.01	10	1000	100	2.931	6.55E-05	4.45E-03	1.51E-04
	20			3.835	1.06E-04		
	40			4.273	9.92E-05		
	10	2000	200	10.363	1.50E-04		
	20			12.683	1.71E-04		
	40			16.047	1.16E-04		
0.001	10	1000	100	27.395	1.74E-04	1.22E-03	1.94E-06
	20			31.758	1.25E-03		
	40			40.7813	4.90E-05		
	10	2000	200	102.216	2.47E-05		
	20			117.819	3.63E-05		
	40			153.327	2.56E-05		

The tests were performed 3 times for each different parameter, including numerical solution. ANN Parameters of the best average Mean Square Error (MSE) found in tests, while still lower than FD's error, are as follow:

TABLE 2. ANN parameters with average MSE of 9.92E-05

Parameter	Value
dt	0.01
time	0.1
total node	41
learning rate	0.01
total neural	40
total epoch	1000
total batch	100

With precede parameters, following graph was created in one of the runs which took 4.422 sec to finish.

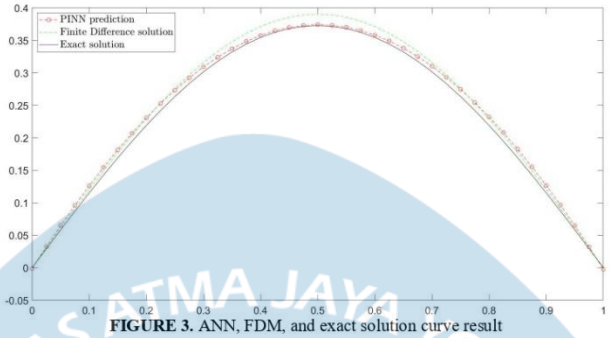

FIGURE 3. ANN, FDM, and exact solution curve result

The MSE of correspondent figure 2 is 6.84E-05, While the boundary error at $y(0)$ is 1.42E-04 and $y(1)$ is 4.29E-06. The weights and biases of ANN is as follows:

TABLE 3. ANN final weights and biases values from table 2 parameters values

Node	w_0	w_1	b_0	b_1
1	-0.3375	-1.7061	1.9830	-1.8786
2	1.5018	-0.8062	-1.0649	
3	-1.4589	-1.1906	-1.4667	
4	-0.9171	1.8972	-0.2306	
5	1.5830	-0.2690	0.2439	
6	1.6474	-0.1372	-0.7406	
7	-1.7842	-1.6888	0.2816	
8	-0.6294	0.6356	1.3591	
9	0.5760	-0.4791	1.8782	
10	-0.0592	0.4046	-0.4381	
11	0.4189	-0.8092	0.8824	
12	-0.9770	1.5457	-0.7300	
13	1.3516	0.8549	-0.0026	
14	1.5459	-1.0052	-1.4968	
15	1.4017	-0.0896	1.4798	
16	0.2302	0.8987	-1.2134	
17	0.7792	-0.3867	1.3685	
18	2.2155	1.3087	0.3756	
19	1.8069	1.0781	1.4894	
20	-2.1035	1.5836	2.4888	
21	-1.7205	-0.7043	-0.5077	
22	1.4368	-0.0260	0.0688	
23	-0.8468	1.6117	0.5377	
24	0.4748	-1.3419	1.7484	
25	-0.9248	-0.0706	1.3504	
26	-1.0011	1.4406	-0.0799	
27	-1.7310	1.0978	1.8970	
28	0.1494	1.7401	-1.5564	
29	0.1295	-0.4292	-0.0062	
30	-0.6355	-1.8451	-0.6807	

31	-0.1305	-0.8844	-0.3207
32	1.3868	1.3706	1.2164
33	-2.1840	-1.2292	0.2071
34	-1.4324	0.2235	-1.8357
35	-0.4574	1.5906	-1.3294
36	-1.6368	-0.3455	-1.1293
37	0.4448	0.4368	-0.0805
38	-0.5452	1.6234	-1.7452
39	-0.3634	-0.9520	-1.8923
40	-1.4658	-0.8377	0.4322

CONCLUSION

ANN has been used to discretize derivatives of space, where time was integrated by Euler's Method. The implementation of ANN as alternative of FD showed promising result as it managed to outperform FD in terms of error in several parameter tunings as showed in previous discussion. Furthermore, ANN managed to predict whole unknown partial differential result, rather than partially. Nevertheless, the lengthy running time of ANN is a major setback.

ACKNOWLEDGMENTS

The work has been financially supported in part by University of Atma Jaya Yogyakarta. Corresponded Matlab programs are available in here [14]. The ANN program writing is influenced heavily by following code [15] from refs [16].

REFERENCES

1. Y. LeCun, Y. Bengio, and G. Hinton, Nature 521, 436-444 (2015).
2. G. E. Karniadakis, Ioannis G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, Nature 3, 422-440 (2021).
3. M. Raissi, P. Perdikaris, and G. E. Karniadakis, J. Comput. Phys. **378**, 686-707 (2019).
4. C. Bajaj, L. McLennaan, T. Andeen, and A. Roy, Mach. Learn.: Sci. Technol. **4**, 015013 (2023).
5. M. R.-Behesht, C. Huber, K. Shukla, and G. E. Karniadakis, J. Geophys. Res. **127**, e2021JB023120 (2022).
6. S. Alkhadhr and M. Almekkawy, J. Sens. **23**, 2792 (2023).
7. N. Zobeiry and K. D. Humfeld, Eng. Appl. Artif. Intell. **101**, 104232 (2021).
8. X. Jin, S. Cai, and G. E. Karniadakis, J. Comput. Phys. **426**, 109951 (2021).
9. H. Elvazi, M. Tahani, P. Schatter, and R. Vinuesa, Phys. Fluids **34**, 075117 (2022).
10. P. Guofei, L. Lu, and G. E. Karniadakis, SIAM J. Sci. Comput. **41**, A2603-A2626 (2019).
11. J. Han, A. Jentzen, and Weinan E, "Solving high-dimensional partial differential equations using deep learning" in *Proceedings of the National Academy of Sciences of the United States of America* (2018), Vol. 115, pp. 8505-8510.
12. C. Song, T. Alkhalifah, and U. B. Waheed, Geophys. J. Int. **225**, 846-659 (2021).
13. O. Noakoasteen, S. Wang, Z. Peng, and C. Christodoulou, IEEE Trans. Antennas Propag. **1**, 9158400 (2020).
14. T. P. Agatho, (2023), available at https://github.com/Theodoret/PINN_1D_HeatTransfer_Euler, accessed on 11 May 2023.
15. A. Almqvist, "Physics-informed neural network solution of 2nd order ODE:s" (2023), available at https://www.mathworks.com/matlabcentral/fileexchange/96852-physics-informed-neural-network-solution-of-2nd-order-ode-s, MATLAB Central File Exchange, retrieved May 3, 2023.
16. A. Almqvist, Lubricants **9**, 82 (2021).

4. Naskah konferensi ICTS 2023

Solving the two-dimensional conduction Heat Transfer Problem by Using Artificial Neural Network method

Theodoret Putra Agatho
Department of Informatics
University of Atma Jaya Yogyakarta
Yogyakarta, Indonesia
190710233@students.uajy.ac.id

Pranowo
Department of Informatics
University of Atma Jaya Yogyakarta
Yogyakarta, Indonesia
pranowo@uajy.ac.id

Andi Wahju Rahardjo Emanuel
Department of Informatics
University of Atma Jaya Yogyakarta
Yogyakarta, Indonesia
andi.emanuel@uajy.ac.id

***Abstract*—In the last decade, Artificial Neural Networks (ANNs) have progress in significant rate and start to meld into various knowledge fields. In modeling and simulation field, Physic-Informed Neural Networks (PINNs), emerge as derivative of ANNs, is being explored extensively its potential. Hence, this paper tried to take a closer look of ANNs on solving the Partial Differential Equations (PDEs) in a simple 2D Heat Transfer problem while the time is integrated by Euler's Method. Later, results were compared to Finite Difference (FD) Method and exact solution. The paper's results show simple ANNs able to solve the problem consistently in limited degree of success.**

Keywords—Heat Transfer, Artificial Neural Network, Partial Differential Equations, Finite Difference Method, and Euler's Method

I. Introduction

In modeling and simulating, there are several techniques to solve field problems, known as Differential Equations (DEs), in the governing equation. In general, these techniques are separated into 2 categories: Analytical Methods (Separation of Variables or Fourier Method, etc.) and Numerical Methods (Finite Difference Method, Finite Volume Method, etc.). Those methods have been applied in fields such as wave and heat transfer [1].

Alternatively, DEs can be solved by Artificial Neural Networks (ANNs) as they share similarity to prior methods, which are approximation functions. Its usage had been advocated in 1997 [2] but was restricted due to the limitations of computational capabilities and neural network applications at the time. This idea resurfaced recently as significant improvements in computation and the introduction of deep learning appeared, which solved restrictions in the preceding neural network.

Even so, one might point out that ANNs still encounter problems, namely being susceptible to overfit and big data requirements, thus favoring the Numerical Methods over them. Which is correct, but by enforcing physic laws into ANNs, prior problems can be minimized while still unlocking the possibility to achieve results in conditions where Numerical Methods are unable to, such as partially known physic laws [3] or the introduction of noise in data [4]. Furthermore, ANNs have the potential to outperform FD in terms of accuracy [5]. This type of ANNs are known widely as Physic-Informed Neural Networks (PINNs). PINNs have been applied in recent cases, i.e., lubricants [6, 7], active cooling [8], traffic state estimation [9], pandemic dispersion [10], and some fluid cases [11].

From the preceding discussion, it's clear that ANNs in simulation and modeling have a well-established basis and are still being explored further. This paper tries to take a better look at ANNs when solving simple 2D Heat Transfer Problems, where ANNs have to predict the entire DEs equation results.

II. Theory Background

Consider the following heat transfer equation, where the thermal conductivity, density, and heat capacity are a constant (α):

$$\frac{\delta u}{\delta t} = \alpha\left(\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2}\right) \tag{1}$$

By applying Euler's implicit method, equation (1) turns into:

$$\frac{u^{n+1} - u^n}{dt} = \frac{\delta u}{\delta t} \tag{2}$$

$$u^{n+1} = u^n + dt\left(\alpha\left(\frac{\delta^2 u^{n+1}}{\delta x^2} + \frac{\delta^2 u^{n+1}}{\delta y^2}\right)\right) \tag{3}$$

A Feedforward Neural Network was presented as the type of ANNs utilized to solve PDEs. This ANN consists of an input layer that receives x and y inputs, a hidden layer composed of N-many neurons with the Sigmoid Function (φ) as the activation function, and an output layer. By regarding ANN as an approximation function, it can be considered a counterpart of the numerical method; hence, it is differentiable, and the differentiations are taken into account in the loss term. Because of this, the considered ANN architecture is as follows:

This research was funded by University of Atma Jaya Yogyakarta

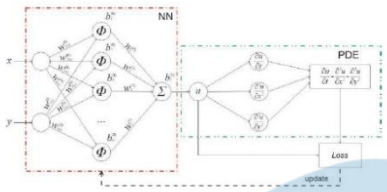

Fig. 1. Purposed ANN architecture

Where the loss function is:

$$l = \langle (u^{n+1} - u^n - dt(\alpha(\frac{\delta^2 u^{n-1}}{\delta x^2} + \frac{\delta^2 u^{n-1}}{\delta y^2})))^2 \rangle + \langle (Bu^{n-1} - B)^2 \rangle \quad (4)$$

Where $\langle f \rangle$ is the mean of f, Bu^{n+1} is the boundary of u^{n+1}, and B denotes the known boundary value. Based on Fig. 1, u^{n+1} can be considered accordingly as the following equation:

$$u^{n+1} = b^{(l)} + \sum w^{(l)} \varphi(x w^{(0)x} + y w^{(0)y} + b^{(0)}) \quad (5)$$

ANN was trained with optimization technique, viz., the computation of gradient error (loss) for each respective network parameter is incorporated to minimize the loss function. Following is a part of the gradient error equation, which in this case is toward a weight.

$$\frac{\delta l}{\delta w} = \langle 2(u^{n+1} - u^n - dt(\alpha(\nabla^2 u^{n-1}))) (\frac{\delta u^{n-1}}{\delta w} - dt(\alpha(\frac{\delta \nabla^2 u^{n-1}}{\delta w}))) \rangle + \langle 2(Bu^{n-1} - B) \frac{\delta B u^{n-1}}{\delta w} \rangle \quad (6)$$

By taking preceding theories into consideration, the program was written as the following flowchart:

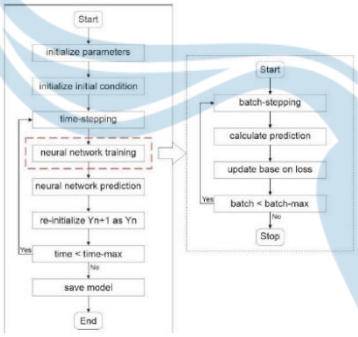

Fig. 2. ANN Program main idea flowchart

III. Results and Discussion

The program was written in MATLAB and run on a computer with an Intel Core i7-9750H CPU. A variety of hyperparameters were tested at least three times for each change, and the results were noted. Results were compared later by their averages of runtime and Mean Square Error (MSE) against the exact results.

The expected results were taken from case problem 3.1 on page 96 in [12]. The boundary and initial values are,

$$0 < x < 4,\ 0 < y < 2,\ u(0,y) = 300 = u(4,y),\ u^0 = 100 \quad (7)$$

The exact solution that accommodates conditions (7) is as follows:

$$u = 300 + 2(100 - 300)\sum_m e^{-\frac{m\pi}{\delta y}^2 \alpha t} \frac{1-(-1)^m}{m\pi} sin(\frac{m\pi y}{\delta y}) \quad (8)$$

Where $\alpha = 0.1$ and y is a matrix coordinate map of y. Because the activation function is a Sigmoid Function (φ), the ANNs prediction needs to be bounded into [0,1], and the boundary and initial values (7) and exact solution (8) need to be modified to accustom the ANNs. The boundary and initial values (7) are modified as follows:

$$0 < x < 4,\ 0 < y < 2,\ u(0,y) = 1 = u(4,y),\ u^0 = 1/3 \quad (9)$$

And the exact solution (8) turns into,

$$u = 1 + 2(\frac{1}{3} - 1)\sum_m e^{-\frac{m\pi}{\delta y}^2 \alpha t} \frac{1-(-1)^m}{m\pi} sin(\frac{m\pi y}{\delta y}) \quad (10)$$

Following (Figs. 1 and 2) are the results of ANN and the Finite Difference Method, where the total time is 1, the total x-node is 21, and the total y-node is 11, while the learning rate of ANN is 0.001.

TABLE I. ANN RUNTIME AND MSE RESULTS

dt	Neural	Epoch	Batch	Neural Network	
				Average runtime (sec)	*Average MSE*
0.1	32	512	128	62.593	5.10E-3
	64			79.882	2.50E-3
	128			130.240	0.1942[a]
	32	1024	256	249.158	2.73E-3
	64			330.080	3.55E-3
	128			523.456	0.1153[a]
	32	2048	512	1012.753	2.73E-3
	64			1325.933	1.23E-3
	128			2122.067	1.04E-2
0.05	32	512	128	124.996	9.63E-3
	64			157.429	5.07E-3
	128			259.125	0.1976[a]
	32	1024	256	498.567	4.97E-3

dt	Neural	Epoch	Batch	Neural Network	
				Average runtime (sec)	*Average MSE*
	64			634.390	2.23E-3
	128			1041.700	0.1514[a]
	32	2048	512	1992.967	2.80E-3
	64			2534.333	4.81E-3
	128			4079.700	2.99E-3

[a] Results failed to generate desirable result or pattern

TABLE II. FINITE DIFFERENCE RUNTIME AND MSE RESULTS

dt	Finite Difference	
	Average runtime (sec)	*Average MSE*
0.1	4.67E-3	1.02E-4
0.05	6.79E-3	3.37E-5

From Fig. 1, the highest average accuracy that ANN achieved is 1.23E-3. The parameters are as follows:

TABLE III. HIGHEST AVERAGE ACCURACY ANN PARAMETERS

Parameter	Value
dt	0.1
time	1
node ()	(21,11)
learning rate	0.001
neural	64
epoch	2048
batch	512

Subsequent is the result of a single ANN with preceding parameters run, while the following are the Finite Difference Method and exact solution results.

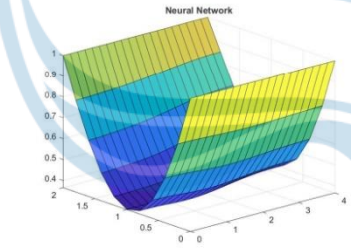

Fig. 3. Neural Network solution result

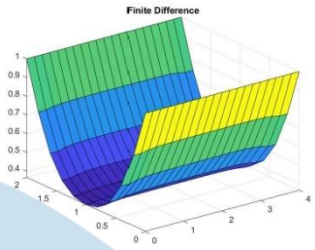

Fig. 4. Finite Difference solution results

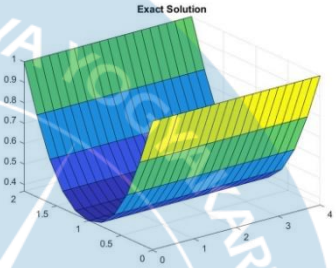

Fig. 5. Exact solution result

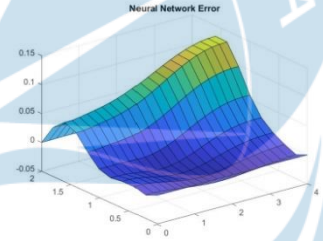

Fig. 6. Neural Network error

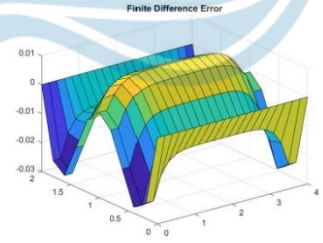

Fig. 7. Finite Difference error

It took 1.38E-3 sec on this run, while the MSE of ANN is 2.40E-3 and its MSE on the boundary is 4.422E-7.

IV. Conclusion

ANNs were used to solve PDEs where a numerical approach such as FD was normally used, while time was integrated by Euler's implicit method. However, in this paper, ANN failed to outperform the accuracy of FD in all different hyperparameter tuning cases but still managed to predict whole PDEs while maintaining low error. This shows promising potential because the architecture of used ANNs is simple compared to current advanced ANNs, while also considering the advantages that has been discussed before. Nevertheless, the computational efficiency problem is a main issue, as indicated by the obtained runtime data.

Acknowledgment

This paper was funded by the University of Atma Jaya Yogyakarta, and the authors expressed gratitude for this. Supplementary materials are available at [13].

References

[1] G. Evans, J. Blackledge, and P. Yardley, Analytic Method for Partial Differential Equations, 2nd Print., Springer :De Montfort, 2001, pp. 50-61.

[2] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," IEEE trans. neural netw., vol. 9, pp. 987-1000, Sept. 1998.

[3] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning", Nat. Rev. Phys., vol. 3, pp. 422-440, May 2021.

[4] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," J. Comput. Phys., vol. 378, pp. 686-707, Feb. 2019.

[5] C. Bajaj, L. McLennan, T. Andeen, and A. Roy, "Recipes for when physics fails: recovering robust learning of physics informed neural networks," Mach. Learn.: Sci. Technol., vol. 4, no. 015013, Feb. 2023.

[6] A. Almqvist, "Fundamentals of physics-informed neural networks applied to solve the reynolds boundary value problem," Lubricants, vol. 9, no. 82, Augst. 2021.

[7] Y. Zhao, L. Guo, and P. P. L. Wong, "Application of physics-informed neural network in the analysis of hydrodynamic lubrication," Friction, Vol. 11, pp. 1253-1264, Sept. 2022.

[8] N. V. Jagtap, M. K. Mudunuru, and K. B. Nakshatrala, "CoolsPINNs: A physics-informed neural network modeling of active cooling in vascular systems," Appl. Math. Model., vol. 122, pp. 265-287, Oct. 2023.

[9] M. Usama, R. Ma, J. Hart, and M. Wojcik, "Physics-Informed Neural Networks (PINNs)-Based Traffic State Estimation: An Application to Traffic Network," Algorithms, vol. 15, no. 447, Nov. 2022.

[10] F. Heldmann, S. Berkhahn, M. Ehrhardt, and K. Klamroth, "PINN training using biobjective optimization: The trade-off between data loss and residual loss," J. Comput. Phys., vol. 488, no. 112211, Sept. 2023.

[11] P. Sharma, W. T. Chung, B. Akoush, and M. Ihme, "A review of physics-informed machine learning in fluid mechanics," Energies, vol. 16, no. 2343, Feb. 2023.

[12] K. A. Hoffmann and S. T. Chiang, Computational Fluid Dynamics, 4th ed., vol. 1, Engineering Education System: Kansas,2000, pp. 96-97.

[13] T. P Agatho, July 2023, available at https://github.com/Theodoret/PINN_2D_HeatTransfer_Euler.