Daftar pustaka

Agus Perdana Windarto, Darmeli Nasution, Anjar Wanto, Frinto Tambunan, Muhammad Said Hasimbuan, Muhammad Noor Hasan Siregar, Muhammad Ridwan Lubis, Solikhun, Yusra Fadhillah, dan Dicky Nofriansyah. 2020. Jaringan Saraf Tiruan Algoritma Prediksi dan Implementasi. Medan. Yayasan Kita Menulis.

Angel Das. 2020. "*Introduction to Artificial Neural Networks for Beginners". Towards Data Science.*

Anil Ananthawamy. 2021. "*Latest Neural Nets Solve World's Hardest Equations Faster Than Ever Before*". *Quanta Magazine.*

Arin Siska Indarwatin. 2019. Penyelesaian Persamaan Diferensial Parsial dengan Pendekatan *Artificial Neural Network*. Universitas Brawijaya.

Danang, A. Pratama, Maharani A. Bakar, N. B. Ismail, dan Mashuri M. 2022. *ANN-based methods for solving partial differential equations*: *a survey. Arab Jurnal of Basic and Applied Sciences.* Vol. 29, No. 1, hal 233-248. *University of Bahrain*.

Eka Pandu Cynthia dan Edi Ismanto. 2017. Jaringan Syaraf Tiruan Algoritma *Backpropagation* dalam Memprediksi Ketersediaan Komoditi Pangan Provinsi Riau. Jurnal Fakultas Sains dan Teknologi UIN Sultan Syarif Khasim Riau. Hal. 271-282. *ISSN*: 2579-5406.

Fadhila Tangguh Admojo dan Yudha Islami Sulistya. 2022. Analisis Performa Algotirma *Stochastic Gradient Descent* (SGD) dalam Mengklasifikasi Tahu Berformalin. *Indonesian journal of Data and Science* (IJODAS). Vol. 3. No. 1. Hal. 1-8. Palembang, Indonesia. ISSN: 2715-9930

Haykin, S. 2009. *Neural Network and Learning machines. Unit State of America: person.*

Ikhwannul Kholis, dan Ahmad Rofii. 2022. Analisis Variasi Parameter *Backpropagation Artificial Neural Network* pada Sistem Pengenalan Wajah Berbasis *principal component analysis. Ejournal* Kajian Teknik Elektro. Vol. 2. No.1. hal. 1-12. Univrsitas 17 Agustus 1945 Jakarta.

Isaac Elias Lagaris, Aristidis Likas, *Member*, IEEE, dan Dimitrios I. Fotiadis. 1998. *Artificial Neural Networks for Solving Ordinary and Partial Differential*

Equations. IEEE Transactions On Neural Network, Vol. 9, No. 5, hal. 988-100.

Jayme Yeremia Wijaya, The Houw Liong, Ken Ratri Retno Wardani. 2007. Perbandingan Penyelesaian Persamaan Differensial Biasa menggunakan Metode Backpropagation, Euler, Heun, dan Rungge-Kutta Orde 4. Jurnal Telematika. 11 (1). Institut Teknologi Harapan Bangsa, Bandung.

Jeffry Kusuma. 2018. Persamaan Differensial Parsial. Makassar. Pusat Kajian Media dan Sumber Belajar Universitas Hasanuddin.

Lebl, J. (2019). Notes on Diffy Qs: Differential Equations for Engineers. Retrieved March 10, 1BC, from https://www.jirka.org/diffyqs/

Muhammad Sabila Haqqi dan Benyamin Kusumoputro. 2022. "Komparasi Metode Optimasi Adam dan SGD dalam *Skema Direct Inverse Control* untuk Sistem Kendali Data Sikap dan Ketinggian Quadcopter". Elkomika: Jurnal Teknik Energi Eletrik, Teknik Telekomunikasi, dan Teknik Elektronika. Vol. 10. N0. 2. Hal. 458-469. Departemen Teknik Elektro, Universitas Indonesia, Indonesia.

Novikaginanto. 2012. https://novikaginanto.wordpress.com/backpropogation. *Backpropagaton*. 15 Februari 2023.

Nugraha, H.G. dan SN, A. 2014. "Optimasi Bobot jaringan Syaraf Tiruan menggunakan Partikel *Swarm Optimization*". IJCCS (*Indonesian Journal of Computing and Cybernetics Systems*). 8 (1), hal. 25-36. Doi:10.22146/ijccs.

Remco Van Der Meer, Cornelis Oosterlee dan Anastasia Borovykh. 2021. *Optimally weight loss for solving PDEs with Neural Networks. CWI Science Park 123, 1098 XG Amsterdam, Netherlands, Delf University of Tehnology Van Mourik Broekmanweg 6, 2628 XE Delf, Neterlands, dan Department of Computing, Imperial College London, London SW7 @AZ, UK.* arXiv:2002.06269v4.

Sigit Kusmaryanto. 2014. Jaringan Saraf Tiruan *Backpropagation* untuk Pengenalan Wajah Metode Ekstraksi Fitur Berbasis Histogram. Jurnal EECCIS Vol. 8, No. 2. Electrical Engineering Universitas Brawijaya.

V.I. Avrutskiy. 2017. *Backpropagation generalized for output derivatives*. Computer science. Cornell University. Doi:10.48550/arXiv:1712.04185

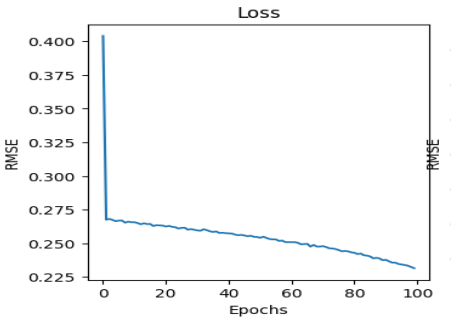
Windarto, A.P., Lubis, M.R. dan Solikhun. 2018. Implementasi JST pada Prediksi Total Laba Rugi Komprehensif Bank Umum Konvensional dengan *Backpropagation*. Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK). 5 (4), hal. 411-418. Doi: 10.25126/jtiik.201854767.

Yuliana Diah Pristanti dan Fredy Windana. 2015. Pengembangan Metode *Neural Network* untuk Menemukan Karakter Seseorang. Jurnal STT STIKMA Internasional. Vol.6, No.1, hal. 9-27.

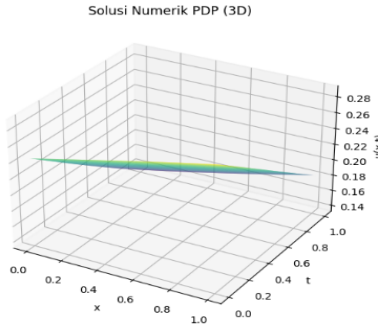
Lampiran

Lampiran 1. Grafik hasil *training*

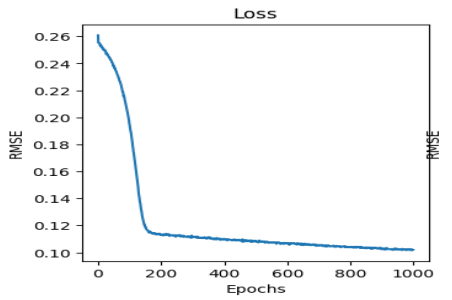
1. Persamaan Gelombang 1-D



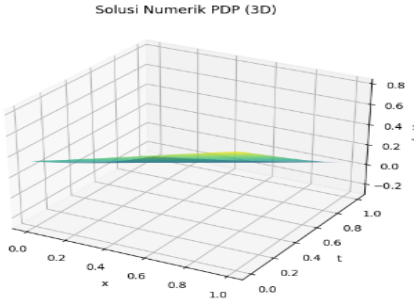
Gambar 1. Perubahan nilai RMSE pada epoch ke-100



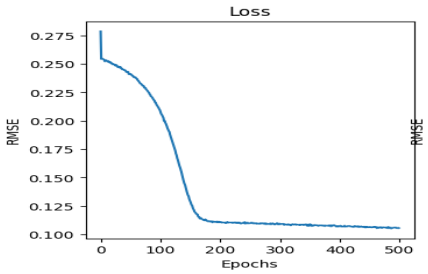
Gambar 2. Grafik Solusi BPNN *epoch* ke-100



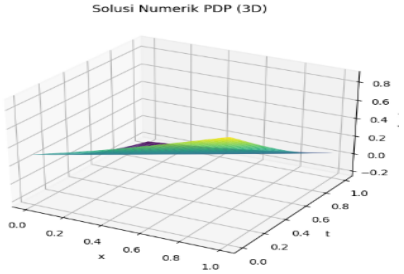
Gambar 3. Perubahan nilai RMSE pada epoch ke-500



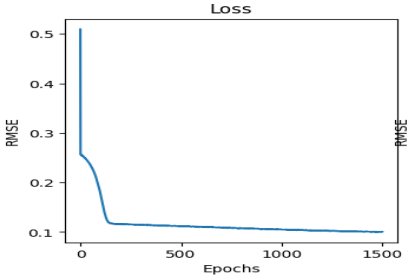
Gambar 4. Grafik Solusi BPNN *epoch* ke-500



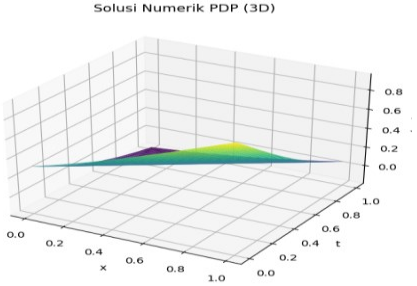
Gambar 5. Perubahan nilai pada epoch ke-1000



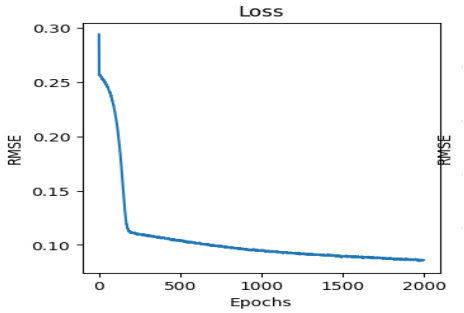
Gambar 6. Grafik Solusi BPNN *epoch* ke-1000



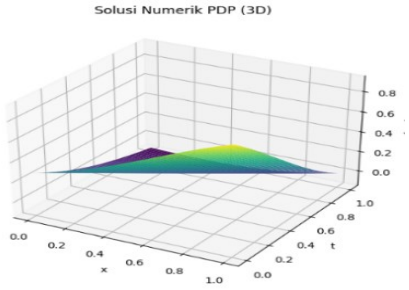
Gambar 7. Perubahan nilai pada epoch ke-1500



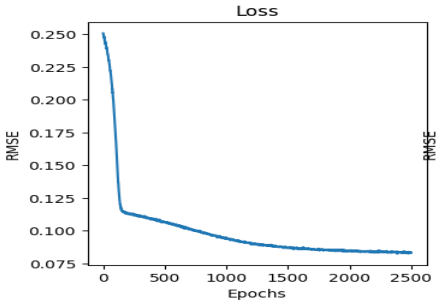
Gambar 8. Grafik Solusi BPNN *epoch* ke-1500



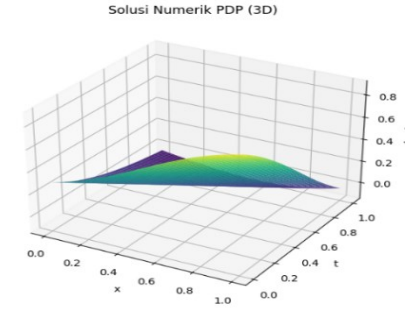
Gambar 9. Perubahan nilai pada epoch ke-2000



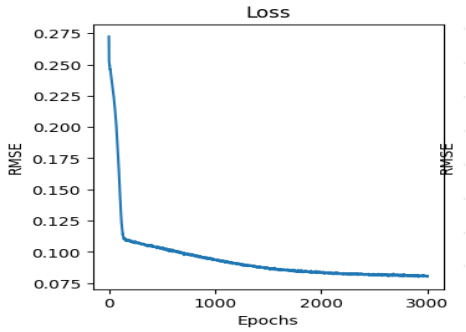
Gambar 10. Grafik Solusi BPNN *epoch* ke-2000



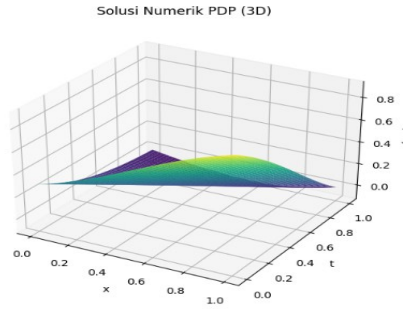
Gambar 11. Perubahan nilai RMSE pada epoch ke-2500



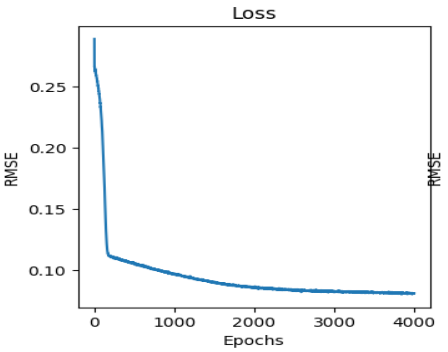
Gambar 12. Grafik Solusi BPNN *epoch* ke-2500



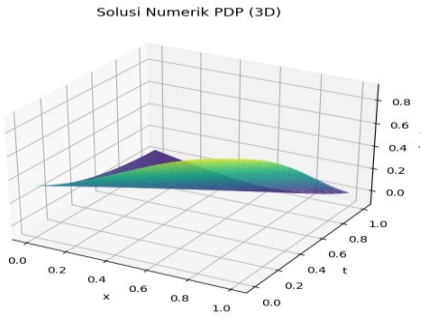
Gambar 13. Perubahan nilai RMSE pada epoch ke-3000



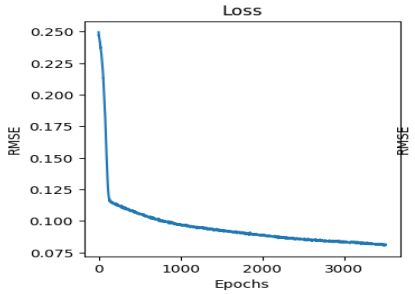
Gambar 14. Grafik Solusi BPNN *epoch* ke-2500



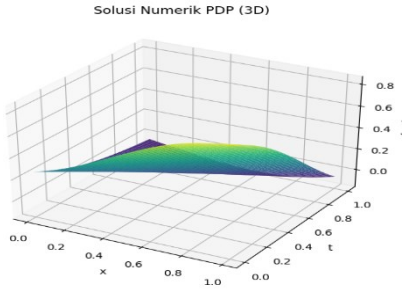
Gambar 17. Perubahan nilai RMSE pada epoch ke-4000



Gambar 18. Grafik Solusi BPNN *epoch* ke-4000

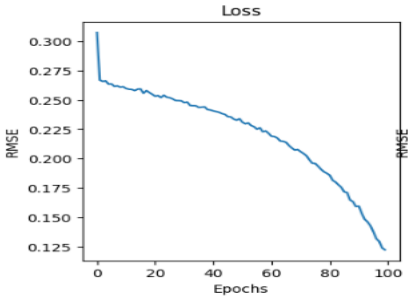


Gambar 19. Perubahan nilai RMSE pada epoch ke-3505

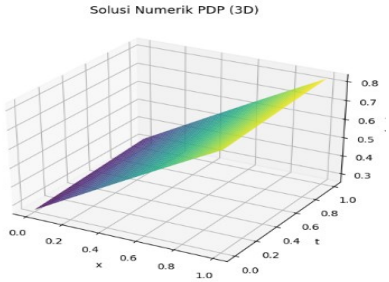


Gambar 20. Grafik Solusi BPNN *epoch* ke-3505

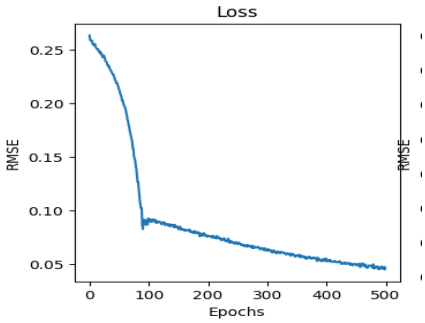
2. Persamaan Burger *inviscid*



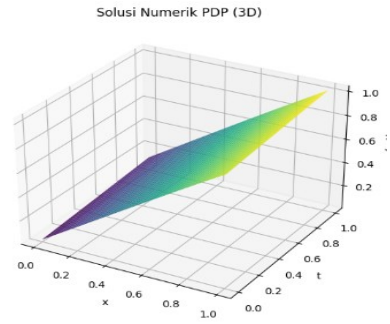
Gambar 21. Perubahan nilai RMSE pada epoch ke-100



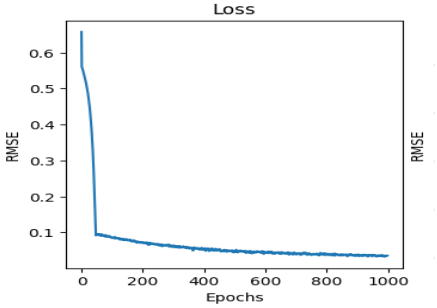
Gambar 22. Grafik Solusi BPNN *epoch* ke-100



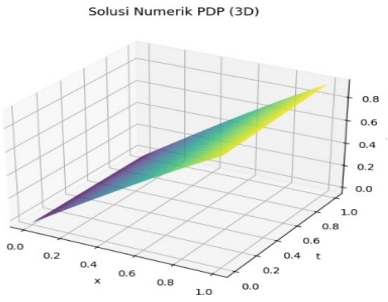
Gambar 23. Perubahan nilai RMSE pada epoch ke-500



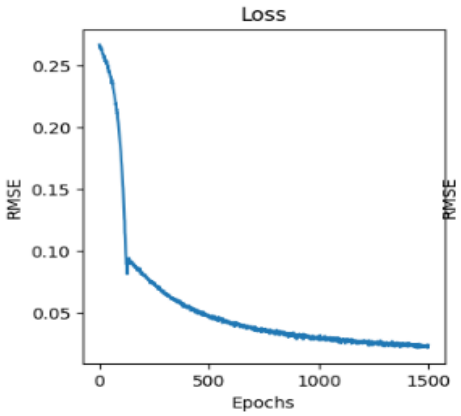
Gambar 24. Grafik Solusi BPNN *epoch* ke-500



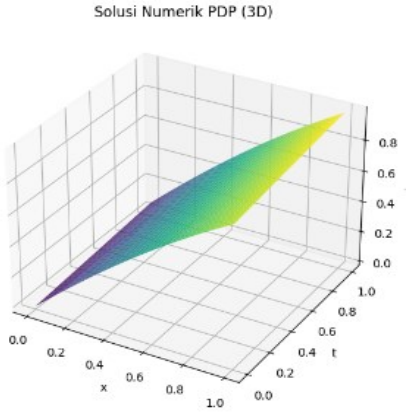
Gambar 25. Perubahan nilai RMSE pada epoch ke-1000



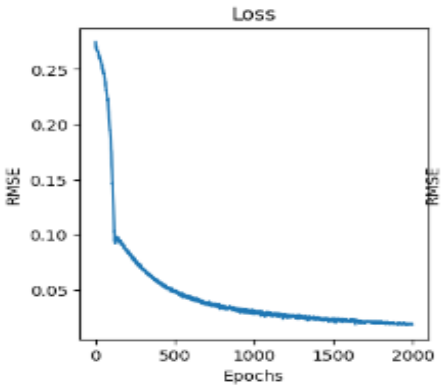
Gambar 26. Grafik Solusi BPNN *epoch* ke-1000



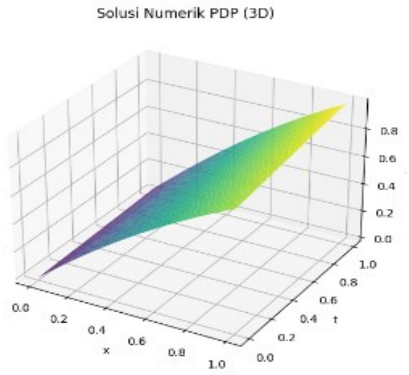
Gambar 27. Perubahan nilai RMSE pada epoch ke-1500



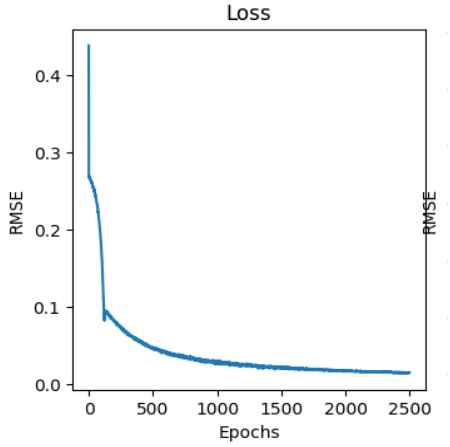
Gambar 28. Grafik Solusi BPNN *epoch* ke-1500



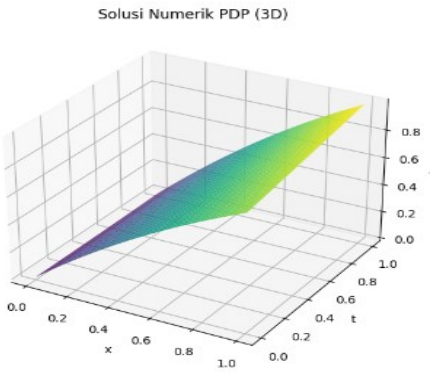
Gambar 29. Perubahan nilai RMSE pada epoch ke-2000



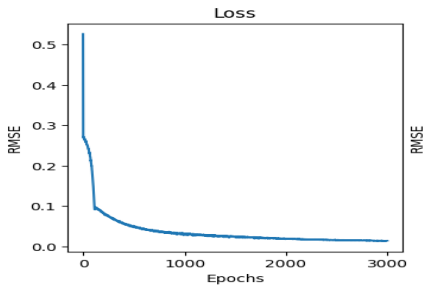
Gambar 30. Grafik Solusi BPNN *epoch* ke-2000



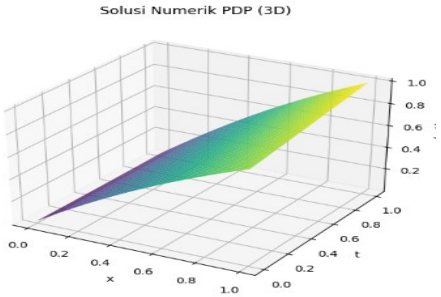
Gambar 31. Perubahan nilai RMSE pada epoch ke-2500



Gambar 32. Grafik Solusi BPNN *epoch*ke-2500

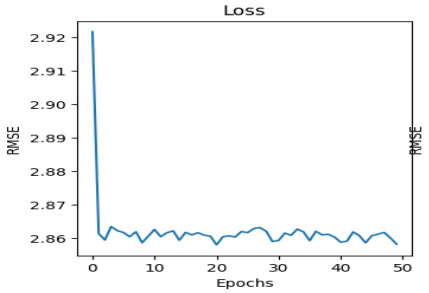


Gambar 33. Perubahan nilai RMSE pada epoch ke-3000

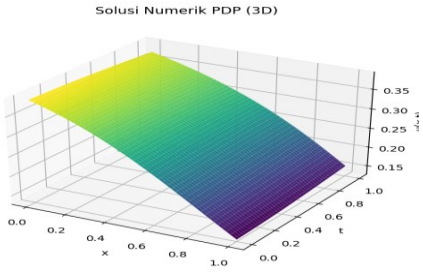


Gambar 34. Grafik Solusi BPNN *epoch* ke-3000

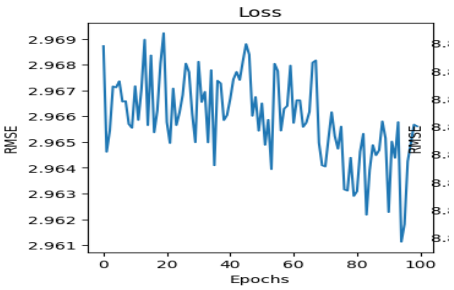
3. Persamann Panas 1-D



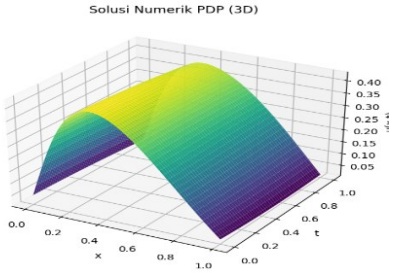
Gambar 35. Perubahan nilai RMSE pada epoch ke-50



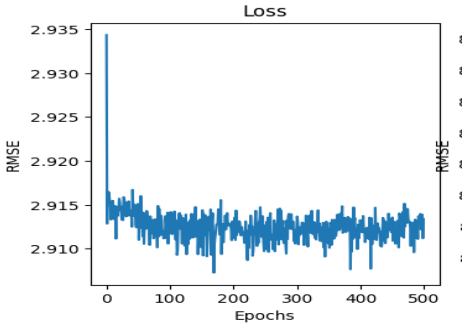
Gambar 36. Grafik Solusi BPNN *epoch* ke-50



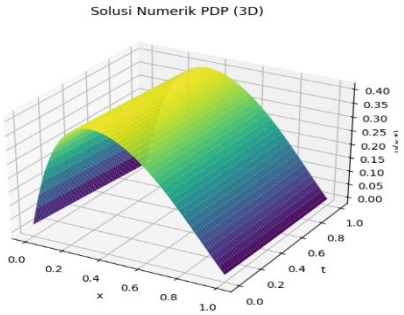
Gambar 37. Perubahan nilai RMSE pada epoch ke-100



Gambar 38. Grafik Solusi BPNN *epoch* ke-100



Gambar 39. Perubahan nilai RMSE pada epoch ke-500



Gambar 40. Grafik Solusi BPNN *epoch* ke-500

Lampiran 2. Kode Program *Pyton*

1. Persamaan Gelombang 1-D

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#mendefinisikan RMSE
 def root_mean_squared_error(u_true, outputs):
    return tf.sqrt(tf.reduce_mean(tf.square(outputs - u_true)))

# Membuat model jaringan saraf tiruan dengan backpropagation
# Solusi PDP yang dicari adalah u(x,t)
def PDE_model():
  # Input layer dengan 2 input, yaitu x dan t
  inputs = tf.keras.layers.Input(shape=(2,))

  # Hitung jumlah neuron dalam layer
  num_neurons = 10

  # Inisialisasi bobot dengan metode Nguyen-Widrow
  initializer = tf.initializers.VarianceScaling(scale=0.7,
mode="fan_avg", distribution="uniform")

  # Hidden layer pertama dengan fungsi aktivasi sigmoid
  p = tf.keras.layers.Dense(num_neurons, activation='sigmoid',
kernel_initializer=initializer)(inputs)
  # Hidden layer kedua dengan fungsi aktivasi sigmoid
  p1 = tf.keras.layers.Dense(num_neurons, activation='sigmoid',
kernel_initializer=initializer)(p)
  # Output layer dengan 1 neuron
  outputs = tf.keras.layers.Dense(1)(p1)

# Membuat model dengan input dan output layer
  Model = tf.keras.models.Model(inputs=inputs, outputs=outputs)

# Mengupdate bobot menggunakan optimizer SGD (Stochastic
Gradient Descent) dengan learning rate 0.01
```

```
    sgd_optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)

    model.compile(optimizer=sgd_optimizer,   # Menggunakan SGD
sebagai optimizer
                  loss=root_mean_squared_error,    #menggunakan
RMSE
                  metrics=[root_mean_squared_error])

    return model

# Fungsi untuk membuat data training
def create_training_data(N=1000, Z=1.0):
    # Membuat data x dan t secara acak
    x = np.random.uniform(low=0.0, high=1.0, size=(N, 1))
    t = np.random.uniform(low=0.0, high=Z, size=(N, 1))
    # Membuat data u(x,t) dengan PDP yang diberikan
    c = 2
    u_true = np.sin(x) * np.cos(c*t)
    # Mengembalikan data x, t, dan u dalam bentuk array numpy
    return np.hstack((x, t)), u_true

 # Fungsi untuk melakukan training model
def train_model(model, outputs, u_true, epochs=3500):
    # Melatih model dengan data training
    history   =   model.fit(outputs,   u_true,   epochs=epochs,
batch_size=32, verbose=0)
    # Mengembalikan model yang telah dilatih dan history dari
pelatihan
    return model, history

def evaluate_model(model, outputs, u_true):
    # Mengevaluasi model dengan data test
    loss = model.evaluate(outputs, u_true, verbose=0)

    return loss

# Membuat plot dari loss
def plot_history(history):
    plt.figure(figsize=(8, 4))
```

```
    plt.plot(history.history['loss'])
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.ylabel('RMSE')
    plt.show()

# Membuat data training
x_train, t_train = create_training_data()

# Membuat model
model = PDE_model()
# Melatih model
model, history = train_model(model, x_train, t_train,
epochs=3500)

# Test loss
x_test, t_test = create_training_data(N=1000)
loss = evaluate_model(model, x_test, t_test)
print(f'Test loss: {loss[0]:.4f}')

#memplot loss
plot_history(history)

#memasukkan nilai parameter x dan t untuk solusi akhir
x = np.linspace(0, 1, 100)
t = np.linspace(0, 1, 100)
y_pred = np.transpose([np.tile(x, len(t)), np.repeat(t,
len(x))])

# menentukan solusi PDP sebagai output akhir
u_pred = model.predict(y_pred)

#melihat nilai solusi di setiap epoch
print(u_pred)

#melihat model arsitektur
model.summary()

#membuat plot 3D solusi persamaan gelombang 1-D dengan BPNN
```

```
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(projection='3d')
x, t = np.meshgrid(x, t)

ax.plot_surface(x, t, u_pred_grid, cmap='viridis')

ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u(x,t)')
ax.set_title('Solusi Numerik PDP (3D)')

plt.show()

# membuat plot solusi analitik persamaan gelombang 1-D
def heat_equation_solution(x, t, c=2):
    return np.sin(x) * np.cos(c*t)

# membuat input nlai x dan t
x = np.linspace(0, 1, 100)
t = np.linspace(0, 1, 100)
x, t = np.meshgrid(x, t)

# menghitung u(x, t) berdasarkan solusi analitik yang diberikan
u = heat_equation_solution(x, t, c=2)

# membuat plot dengan 3D
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

# warna Plot
ax.plot_surface(x, t, u, cmap='viridis')

# membuat label
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u(x, t)')

plt.show()
```

2. Persamaan Burger *Inviscid*

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#mendefinisikan RMSE
 def root_mean_squared_error(u_true, outputs):
    return tf.sqrt(tf.reduce_mean(tf.square(outputs - u_true)))

# Membuat model jaringan saraf tiruan dengan backpropagation
# Solusi PDP yang dicari adalah u(x,t)
def PDE_model():
  # Input layer dengan 2 input, yaitu x dan t
  inputs = tf.keras.layers.Input(shape=(2,))

  # Hitung jumlah neuron dalam layer
  num_neurons = 10

  # Inisialisasi bobot dengan metode Nguyen-Widrow
  initializer    =    tf.initializers.VarianceScaling(scale=0.7,
mode="fan_avg", distribution="uniform")

  # Hidden layer pertama dengan fungsi aktivasi sigmoid
  p  =  tf.keras.layers.Dense(num_neurons,  activation='sigmoid',
kernel_initializer=initializer)(inputs)
  # Hidden layer kedua dengan fungsi aktivasi sigmoid
  p1 = tf.keras.layers.Dense(num_neurons, activation='sigmoid',
kernel_initializer=initializer)(p)
  # Output layer dengan 1 neuron
  outputs = tf.keras.layers.Dense(1)(p1)

# Membuat model dengan input dan output layer
  Model = tf.keras.models.Model(inputs=inputs, outputs=outputs)

#  Mengupdate  bobot  menggunakan  optimizer    SGD  (Stochastic
Gradient Descent) dengan learning rate 0.01
    sgd_optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)
```

```
    model.compile(optimizer=sgd_optimizer,    # Menggunakan SGD
sebagai optimizer
                  loss=root_mean_squared_error,     #menggunakan
RMSE
                  metrics=[root_mean_squared_error])

    return model

# Fungsi untuk membuat data training
def create_training_data(N=1000, Z=1.0):
    # Membuat data x dan t secara acak
    x = np.random.uniform(low=0.0, high=1.0, size=(N, 1))
    t = np.random.uniform(low=0.0, high=Z, size=(N, 1))
    # Membuat data u(x,t) dengan PDP yang diberikan
    k = 0.2
    u_true = np.sin(x) * ((k*(t**2))+1)
    # Mengembalikan data x, t, dan u dalam bentuk array numpy
    return np.hstack((x, t)), u_true

 # Fungsi untuk melakukan training model
def train_model(model, outputs, u_true, epochs=2510):
    # Melatih model dengan data training
    history    =    model.fit(outputs,    u_true,    epochs=epochs,
batch_size=32, verbose=0)
    # Mengembalikan model yang telah dilatih dan history dari
pelatihan
    return model, history

def evaluate_model(model, outputs, u_true):
    # Mengevaluasi model dengan data test
    loss = model.evaluate(outputs, u_true, verbose=0)

    return loss

# Membuat plot dari loss
def plot_history(history):
    plt.figure(figsize=(8, 4))
    plt.plot(history.history['loss'])
    plt.title('Loss')
```

```
    plt.xlabel('Epochs')
    plt.ylabel('RMSE')
    plt.show()

# Membuat data training
x_train, t_train = create_training_data()

# Membuat model
model = PDE_model()
# Melatih model
model, history = train_model(model, x_train, t_train,
epochs=2510)

# Test loss
x_test, t_test = create_training_data(N=1000)
loss = evaluate_model(model, x_test, t_test)
print(f'Test loss: {loss[0]:.4f}')

#memplot loss
plot_history(history)

#memasukkan nilai parameter x dan t untuk solusi akhir
x = np.linspace(0, 1, 100)
t = np.linspace(0, 1, 100)
y_pred = np.transpose([np.tile(x, len(t)), np.repeat(t,
len(x))])

# menentukan solusi PDP sebagai output akhir
u_pred = model.predict(y_pred)

#melihat nilai solusi di setiap epoch
print(u_pred)

#melihat model arsitektur
model.summary()

#membuat plot 3D solusi persamaan burger inviscid dengan BPNN
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(projection='3d')
```

```
x, t = np.meshgrid(x, t)

ax.plot_surface(x, t, u_pred_grid, cmap='viridis')

ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u(x,t)')
ax.set_title('Solusi Numerik PDP (3D)')

plt.show()

# membuat plot solusi analitik persamaan burger inviscid
def heat_equation_solution(x, t, k=0.2):
    return np.sin(x) * ((k*(t**2))+1)

# membuat input nlai x dan t
x = np.linspace(0, 1, 100)
t = np.linspace(0, 1, 100)
x, t = np.meshgrid(x, t)

# menghitung u(x, t) berdasarkan solusi analitik yang diberikan
u = heat_equation_solution(x, t, k=0.2)

# membuat plot dengan 3D
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

# warna Plot
ax.plot_surface(x, t, u, cmap='viridis')

# membuat label
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u(x, t)')

plt.show()
```

3. Persamaan Panas 1-D

```
import tensorflow as tf
import numpy as np
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#mendefinisikan RMSE
 def root_mean_squared_error(u_true, outputs):
    return tf.sqrt(tf.reduce_mean(tf.square(outputs - u_true)))

# Membuat model jaringan saraf tiruan dengan backpropagation
# Solusi PDP yang dicari adalah u(x,t)
def PDE_model():
  # Input layer dengan 2 input, yaitu x dan t
  inputs = tf.keras.layers.Input(shape=(2,))

  # Hitung jumlah neuron dalam layer
  num_neurons = 10

  # Inisialisasi bobot dengan metode Nguyen-Widrow
  initializer    =    tf.initializers.VarianceScaling(scale=0.7,
mode="fan_avg", distribution="uniform")

  # Hidden layer pertama dengan fungsi aktivasi sigmoid
  p  =  tf.keras.layers.Dense(num_neurons,  activation='sigmoid',
kernel_initializer=initializer)(inputs)
  # Hidden layer kedua dengan fungsi aktivasi sigmoid
  p1 = tf.keras.layers.Dense(num_neurons, activation='sigmoid',
kernel_initializer=initializer)(p)
  # Output layer dengan 1 neuron
  outputs = tf.keras.layers.Dense(1)(p1)

# Membuat model dengan input dan output layer
  Model = tf.keras.models.Model(inputs=inputs, outputs=outputs)

#  Mengupdate  bobot  menggunakan  optimizer    SGD  (Stochastic
Gradient Descent) dengan learning rate 0.01
    sgd_optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)

    model.compile(optimizer=sgd_optimizer,    # Menggunakan  SGD
sebagai optimizer
```

```
                    loss=root_mean_squared_error,     #menggunakan
RMSE
                    metrics=[root_mean_squared_error])

    return model

# Fungsi untuk membuat data training
def create_training_data(N=1000, Z=1.0):
    # Membuat data x dan t secara acak
    x = np.random.uniform(low=0.0, high=1.0, size=(N, 1))
    t = np.random.uniform(low=0.0, high=Z, size=(N, 1))
    # Membuat data u(x,t) dengan PDP yang diberikan
    n_values = np. arange (1, 101)
    u_true = (400 / ((np.pi**2) * n_values**2))*(np.sin(n_values
* np.pi * x))*(np.exp(-np.pi**2* n_values**2 * 0.003 * t))
    # Mengembalikan data x, t, dan u dalam bentuk array numpy
    return np.hstack((x, t)), u_true

 # Fungsi untuk melakukan training model
def train_model(model, outputs, u_true, epochs=700):
    # Melatih model dengan data training
    history    =    model.fit(outputs,    u_true,    epochs=epochs,
batch_size=32, verbose=0)
    # Mengembalikan model yang telah dilatih dan history dari
pelatihan
    return model, history

def evaluate_model(model, outputs, u_true):
    # Mengevaluasi model dengan data test
    loss = model.evaluate(outputs, u_true, verbose=0)

    return loss

# Membuat plot dari loss
def plot_history(history):
    plt.figure(figsize=(8, 4))
    plt.plot(history.history['loss'])
    plt.title('Loss')
    plt.xlabel('Epochs')
```

```
    plt.ylabel('RMSE')
    plt.show()

# Membuat data training
x_train, t_train = create_training_data()

# Membuat model
model = PDE_model()
# Melatih model
model, history = train_model(model, x_train, t_train,
epochs=700)

# Test loss
x_test, t_test = create_training_data(N=1000)
loss = evaluate_model(model, x_test, t_test)
print(f'Test loss: {loss[0]:.4f}')

#memplot loss
plot_history(history)

#memasukkan nilai parameter x dan t untuk solusi akhir
x = np.linspace(0, 1, 100)
t = np.linspace(0, 1, 100)
y_pred = np.transpose([np.tile(x, len(t)), np.repeat(t,
len(x))])

# menentukan solusi PDP sebagai output akhir
u_pred = model.predict(y_pred)

#melihat nilai solusi di setiap epoch
print(u_pred)

#melihat model arsitektur
model.summary()

#membuat plot 3D solusi persamaan panas 1-D dengan BPNN
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(projection='3d')
x, t = np.meshgrid(x, t)
```

```
ax.plot_surface(x, t, u_pred_grid, cmap='viridis')

ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u(x,t)')
ax.set_title('Solusi Numerik PDP (3D)')

plt.show()

# membuat plot solusi analitik persamaan panas 1-D
def generate_u_values(x, t):
    n_values = np. arange (1, 101)
    u = np.zeros ((len(x),len(t)))
    for n in n_values:
         u +=(400 / ((np.pi**2) * n_values**2))*(np.sin(n_values
* np.pi * x))*(np.exp(-np.pi**2* n_values**2 * 0.003 * t))
    return u

# membuat input nlai x dan t
x = np.linspace(0, 1, 100)
t = np.linspace(0, 1, 100)
x, t = np.meshgrid(x, t)

# menghitung u(x, t) berdasarkan solusi analitik yang diberikan
u = generate_u_values(x, t)

# membuat plot dengan 3D
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

# warna Plot
ax.plot_surface(x, t, u, cmap='viridis')
# membuat label
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u(x, t)')

plt.show()
```