PENYELESAIAN PERSAMAAN DIFERENSIAL PARSIAL DENGAN PENDEKATAN ARTIFICIAL NEURAL NETWORK

SKRIPSI

Oleh:

ARIN SISKA INDARWATIN
155090301111001

JURUSAN FISIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG
2019

Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

UNIVERSITAS BRAWIJAYA REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA REPOSITORY.UB.AC.ID
Repository Universitas Brawijaya

PENYELESAIAN PERSAMAAN DIFERENSIAL PARSIAL DENGAN PENDEKATAN ARTIFICIAL NEURAL NETWORK

SKRIPSI

Sebagai salah satu syarat untuk memperoleh
gelar Sarjana Sains dalam bidang fisika

Oleh:

ARIN SISKA INDARWATIN

155090301111001

JURUSAN FISIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

MALANG

2019

Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

Repository Universitas Brawijaya

LEMBAR PENGESAHAN SKRIPSI

PENYELASAIAN PERSAMAAN DIFERENSIAL PARSIAL DENGAN PENDEKATAN ARTIFICIAL NEURAL NETWORK

Oleh:

ARIN SISKA INDARWATIN
155090301111001

Setelah dipertahankan di depan Majelis Penguji
pada tanggal 18 JUN 2019
Dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana Sains dalam bidang Fisika

Pembimbing I	Pembimbing II
(Eng. Agus Naba, S.Si.,MT)	(Dr. rer. nat. Abdurrouf, S.Si., M.Si.
NIP. 197208061995121001	NIP. 197209031994121001

Mengetahui,
Ketua Jurusan Fisika

(Prof. Dr.rer.nat. Muhammad Nurhuda)
NIP. 196409101990021001

Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

Repository Universitas Brawijaya

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini:

Nama :Arin Siska Indarwatin
NIM :155090301111001
Jurusan :Fisika
Penulis Skripsi Berjudul :

PENYELESAIAN PERSAMAAN DIFERENSIAL PARSIAL DENGAN PENDEKATAN ARTIFICIAL NEURAL NETWORK

Dengan ini menyatakan bahwa :

1. **Isi dari Skripsi saya yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam Skripsi ini.**
2. **Apabila dikemudian hari ternyata Skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 25 April 2019
Yang Menyatakan,

(Arin Siska Indarwatin)
NIM. 155090301111001

Repository Universitas Brawijaya

Repository Universitas Brawijaya

PENYELESAIAN PERSAMAAN DIFERENSIAL PARSIAL DENGAN PENDEKATAN ARTIFICIAL NEURAL NETWORK

ABSTRAK

enyelesaian persamaan diferensial parsial merupakan model matematis ang diterapkan untuk menyelesaikan kasus fisis yang didasarkan pada rinsip atau hukum tertentu yang menyatakan keadaan alami suatu materi. ada penelitian ini, dibuat suatu model yang digunakan untuk menentukan olusi persamaan diferensial parsial dengan pendekatan *artificial neural etwork*. Ide ini didasari dari konsep *feedforward neural network* yang emetakan fungsi tertentu secara maju tanpa ada koneksi *feedback*. Model ni dibagi menjadi dua bagian yaitu bagian fungsi yang menghitung masalah yarat batas dan keadaan awal sistem dengan parameter tetap, dan bagian odel neural network dengan pengaturan parameter bobot yang diubah-bah. Dalam penelitian ini, dilakukan penghitungan kasus persamaan *aplace* dan persamaan difusi dengan syarat batas Dirichlet dan Neumann, engan arsitektur *single layer feedforward neural network* yang berisi 10 euron pada *hidden* layernya. Kemudian hasil pendekatan neural network ibandingkan dengan solusi eksak dan solusi numerik biasa. Dari hasil erbandingan tiap model, penyelesaian persamaan diferensial parsial emiliki akurasi yang mendekati solusi numerik, sehingga dapat dijadikan ebagai alternatif penyelesaian persamaan diferensial pada kasus syarat batas irichlet.

Kata Kunci: *Feedforward neural network*, persamaan diferensial parsial, yarat batas, Persamaan *Laplace*, Persamaan Difusi

Repository Universitas Brawijaya

SOLVING PARTIAL DIFFERENTIAL EQUATION WITH ARTIFIAL NEURAL NETWORK APPROXIMATION

ABSTRACT

olving Partial Differential Equation (PDE) is a mathematics modelling hich is applied to solve physics cases based on natural principles henomena and the laws regarding the property of matters. This study was onducted to get a model to determine the solution of partial differential quation with artificial neural network approximation. The notion of this tudy was based on the concept of feedforward neural network, mapping any unction in forward without any feedback connection. The model was ivided into two parts, the first part satisfies the boundary condition and the nitial conditions of the problem, while the second part contains the neural etwork architecture with adjustable weight parameter which was onstructed to approximate the solution of PDE. This study presented the nethod to solve Laplace equation and diffusion equation in the case of Dirichlet and Neumann boundary conditions. The solution was modeled by onstructing a single layer feedforward neural network with 10 neurons in a idden layer. Then, the result of PDE approximation was compared with xact solution and ordinary numerical method. From the comparison of each nodel, it was concluded that solving PDE with artificial neural network can e an alternative to determine the solution through its accuracy as it is omparable with exact solution in Dirichlet boundary problem.

Keywords: Feedforward neural network, Different partial equation, Boundary conditions, Laplace equation, Diffusion Equation

Repository Universitas Brawijaya

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Allah SWT atas limpahan ahmat dan karunia-Nya sehingga penulis dapat menyelesaikan proposal kripsi yang berjudul **"PENYELESAIAN PERSAMAAN IFERENSIAL PARSIAL DENGAN PENDEKATAN ARTIFICIAL EURAL NETWORK"**. Oleh karena itu, penulis mengucapkan rasa ormat dan terimakasih kepada:

1. Allah SWT atas nikmat dan karunia-Nya yang tak pernah putus sebagai pemilik ilmu.
2. Kedua orang tua penulis yang selalu mengingatkan *deadline*, dan memberi doa serta motivasi.
3. Dr. Eng. Agus Naba,S.Si.,MT, sebagai dosen pembimbing I dan Dr. rer. Nat. Abdurrouf, S.Si. M.Si. yang telah memberikan pengarahan dan masukan kepada penulis selama penyusunan skripsi.
4. Prof.Dr.rer.nat.Muhammad Nurhuda selaku ketua jurusan Fisika Universitas Brawijaya.
5. Dr. Mauludi Ariesto Pamungkas, S.Si., M.Si. sebagai Kepala Laboratorium Simulasi dan Pemodelan yang telah memberikan izin untuk menggunakan fasilitas laboratorium, sekaligus sebagai penguji.
6. Bapak dan Ibu dosen, laboran, serta karyawan jurusan Fisika atas didikan dan bantuan selama proses perkuliahan.
7. Teman-teman Tabook Indonesia, yang memberikan support pada penyelesaian skripsi ini.
8. Teman-teman di Jurusan Fisika dan kolega di laboratorium Simulasi dan Pemodelan yang memberikan bantuan pada penulis.
9. Pembaca yang telah meluangkan waktunya untuk membaca laporan skripsi ini.

Penulis memohon maaf apabila didapati kekurangan dalam penyusunan kripsi ini. Diharapkan skripsi ini dapat bermanfaat bagi kita semua.

Penulis

Repository Universitas Brawijaya

DAFTAR ISI

Halaman

HALAMAN JUDUL i

LEMBAR PENGESAHAN SKRIPSI iii

LEMBAR PERNYATAAN v

KATA PENGANTAR xi

DAFTAR ISI xiii

DAFTAR GAMBAR xvii

DAFTAR LAMPIRAN xix

DAFTAR TABEL xx

BAB I PENDAHULUAN 1

1.1 Latar Belakang 1

1.2 Rumusan Masalah 3

1.3 Batasan Masalah 4

1.4 Tujuan 4

1.5 Manfaat 4

BAB II TINJAUAN PUSTAKA 5

2.1 Persamaan Diferensial 5

2.2 Contoh Kasus Persamaan Diferensial Parsial 7

2.2.1 Persamaan *Laplace* 7

2.2.2 Persamaan Difusi 8

2.3 Kondisi Awal dan Syarat Batas pada Persamaan Diferensial 8

2.3.1 Syarat batas *Dirichlet* 9

2.3.2 Syarat batas *Neumann* 9

2.3.3 Syarat batas *Robin* 10

Repository Universitas Brawijaya

2.3.4 Syarat batas campuran 10
2.4 Metode Numerik Penyelesaian Persamaan Diferensial Parsial 10
2.5 Artificial Neural Network 12
2.5.1 Model Artificial Neural Network 14
2.5.2 Feed Forward Neural Network 16
2.5.3 Stochastic Gradient Descent (SGD) 17
2.5.4 Supervised Learning 18
2.6 Metrik Evaluasi 18
BAB III METODE PENELITIAN 21
3.1 Waktu dan Tempat 21
3.2 Peralatan dan Bahan 21
3.3 Prosedur Penelitian 21
3.3.1 Persiapan komputasi 21
3.3.2 Pembuatan Model Solusi Umum 22
3.3.3 *Pre-processing* Data 23
3.3.4 Training *Neural Network* 25
3.3.5 Testing 28
BAB IV Hasil dan Pembahasan 29
4.1 Model Neural Network dan Penentuan Syarat Batas 29
4.1.1 Persamaan *Laplace* (Syarat Batas *Dirichlet*) 31
4.1.2 Persamaan *Laplace* (Syarat Batas *Neumann*) 34
4.1.3 Persamaan Difusi 1 Dimensi (Syarat Batas Dirichlet) 37
4.1.4 Persamaan Difusi 2 Dimensi 40
4.2 Perbandingan Perhitungan Persamaan Diferensial Parsial Dengan Neural Network Terhadap Hasil Analitik 45

4.2.1 Kasus 1 (Persamaan Laplace Syarat Batas Dirichlet) ..45
4.2.2 Kasus 2 (Persamaan Laplace Syarat Batas Neumann) 45
4.2.3 Kasus 3 (Persamaan Difusi 1D Syarat Batas Dirichlet)46
4.2.4 Kasus 4 (Persamaan Difusi 2D syarat Batas Dirichlet)47
AB V PENUTUP 49
5.1 Kesimpulan 49
5.2 Saran 49
AFTAR PUSTAKA 51
ampiran 1. Grafik Visual Hasil Training 53
Interpolasi hasil training untuk kasus ke 1 53
Interpolasi hasil training untuk kasus ke 2 54
Interpolasi hasil training untuk kasus ke 3 55
ampiran 2. Kode Program 57
Kode program Kasus 1 57
Kode program Kasus 2 63
Kode Program Kasus 3 67
Kode Program Kasus 4 72

Repository Universitas Brawijaya

DAFTAR GAMBAR

Halaman

Gambar 2.1 Domain dan syarat batas pada sistem 8
Gambar 2.2 Neuron pada saraf biologi 13
Gambar 2.3 Skema neuron biologi dan neuron pada ANN 13
Gambar 2.4 Arsitektur ANN sederhana 14
Gambar 2.5. Representasi bagian input, hidden layer, dan output ANN 15
Gambar 3.1 Data flow pada proses *training* 24
Gambar 3.2 *Grafik fungsi aktivasi sigmoid dan tanh* 25
Gambar 3.3 Bagan kerja *Neural Network* 26
Gambar 4.1 Sketsa kasus persamaan (4.9) 31
Gambar 4.2. Hubungan epoch terhadap loss (kasus 1) 32
Gambar 4.3. Grafik visualisasi hasil prediksi ANN kasus 1 33
Gambar 4.4. Grafik visualisasi hasil perhitungan analitik kasus 1 33
Gambar 4.5. Sketsa kasus 2 34
Gambar 4.6. Hubungan epoch terhadap loss kasus 2 35
Gambar 4.7. Grafik hasil visualisai prediksi ANN kasus 2 36
Gambar 4.8. Grafik hasil perhitungan analitik kasus 2 36
Gambar 4.9 Sketsa kasus 3 37
Gambar 4.10. Hubungan epoch terhadap loss kasus 3 38
Gambar 4.11. Grafik visualisasi hasil prediksi ANN kasus 3 39

Gambar 4.12. Grafik visualisasi hasil perhitungan analitik kasus 3 39
Gambar 4.13. Sketsa kasus 4 40
Gambar 4.14. Grafik visualisasi prediksi ANN kasus 4 42
Gambar 4.15. Grafik visualisasi hasil perhitungan analitik kasus 4 42
Gambar 4.16. Hubungan *epoch* terhadap loss kasus 4 43
Gambar 4.17. Grafik nilai loss terhadap banyaknya epoch kasus ke-4 43
Gambar 4.19 Visualisasi prediksi difusi pada t=4 44
Gambar 4.20 Visualisasi prediksi difusi pada t=5 44

DAFTAR LAMPIRAN

Halaman

ampiran 1 Grafik interpolasi hasil training.....43

ampiran 2 Kode program 57

DAFTAR TABEL

abel 4.1Perbandingan solusi ANN dengan Numerik kasus 1 45
abel 4.2 Perbandingan solusi ANN dengan Numerik kasus 2 46
abel 4.3. Perbandingan solusi ANN dengan Numerik kasus 3 46
abel 4.4 Perbandingan solusi ANN dengan Numerik kasus 4 47

BAB I
PENDAHULUAN

1.1 Latar Belakang

Solusi numerik dari persamaan diferensial biasa dan persamaan diferensial parsial dapat dihitung secara konvensional dengan pendekatan *finite element, finite volume,* dan *finite difference* dengan memperhatikan diskritisasi domainnya. Solusi pendekatan yang dilakukan pada FEM dan FVM adalah dengan mencacah sejumlah titik pada bagian persamaan diferensialnya terhadap domain ruang dan waktu. Dari hasil diskritisasi tersebut akan diperoleh sistem linear aljabar yang dinyatakan dalam suatu operasi matriks. Pada proses komputasinya, tiap langkah pada domain ruang dinyatakan dalam skema *grid/mesh*, yang mana pada proses ini membutuhkan *computation cost* cukup besar. Solusi lain yang lebih sederhana yaitu dengan FTCS (*forward time central space)* yaitu menyelesaikan persamaan diferensial secara diskrit dengan mencacah berdasarkan penurunan secara maju pada domain waktu dan penurunan di tengah pada domain ruangnya, akan tetapi solusi ini sering menghasilkan nilai *error* yang besar karena hasil *truncation*.

Dalam *artificial neural network*, terdapat ide untuk melakukan pemetaan suatu fungsi di konsep *classifier* yaitu pengklasifikasian objek dengan memanfaatkan persamaan linear , $y = f*(x)$ untuk meregresi data (Goodfellow, Bengio, & Courville, 2016). Jika algoritma pemetaan fungsi dapat dilakukan dengan *neural network*, maka penghitungan suatu fungsi tertentu akan mungkin diselesaikan dengan algoritma neural network.

Artificial Neural Network merupakan bagian dari algoritma *machine learning* yaitu algoritma yang dapat belajar melalui data. Dalam hal ini algoritma *machine learning* akan mengerjakan sutau program yang akan belajar dari data sebelumnya berdasarakan perintah yang diberikan, kemudian performa tersebut diukur untuk mengimprovisasi proses belajar dari program tersebut. Dalam bidang data analitik atau data *science*, beberapa algoritma *machine learning*

apat digunakan untuk membantu dalam proses simulasi dan emodelan.

Ide pengembangan *neural network* selanjutnya adalah engaplikasian algoritma tersebut untuk menganalisa sistem yang lebih ompleks seperti proses fisikal, biologi, maupun pada sistem eteknikan. Dalam beberapa penghitungan sistem fisis, sering dihadapi asalah penyimpulan hasil dari satu perhitungan dari data akuisisi yang ersifat parsial. Hal tersebut mendasari bagaimana algoritma *machine earning* dapat diterapkan pada komputasi fisis untuk mendapatkan esimpulan yang tepat dari data parsial yang tersedia. *Neural network* kan ditraining untuk menyelesaikan kasus *supervised learning* yang esuai dengan hukum fisika yang dinyatakan dalam persamaan umum on-linear diferensial parsial.

Untuk menyelesaikan permasalahan model persamaan yang ederhana, dapat dilakukan dengan membuat *multilayer percepton*, aitu *neural network* sederhana tanpa melibatkan *hidden layer* Zaccone, 2016). Bentuk lain dari pendeketann berikutnya yaitu dengan emanfaatkan arsitektur *feedforward neural network* untuk enyelesaikan persamaan diferensial sehingga didapatkan pendekatan enghitungan yang mudah dan akurat (Chiaramonte & Kiener, 2013; layati & Karami, 2007).

Dengan menerapkan arsitektur *neural network*, diharapkan dapat engatasi masalah non-linear tanpa menentukan asumsi sebelumnya, nearisasi, atau pengaturan diskritisasi *time-step* pada penyelesaian ersamaan diferensial parsial secara numerik. Paradigma metode ini erupakan paradigma baru dalam dunia *modeling* dan komputasi yang iperkaya dengan kerangka berpikir *neural network* dan automasi di erkembangan teori fisika matematika. Pada penelitian sebelumnya, enggabungan konsep *neural network* untuk pemecahan kasus fisis iterapkan pada konteks *Physics Informed Neural Network* yaitu iklasifikasikan pada dua masalah utama yaitu: solusi berdasarkan data *data-driven solution*) (Yeo & Melnyk, 2019) dan penelitian bedasarkan ata (*data-driven discovery*) pada penyelesaian persamaan diferensial

(Raissi, Perdikaris, & Karniadakis, 2017). Kemudian, pada perkembangan berikutnya diterapkan metode yang serupa untuk penyelesaian persamaan diferensial secara *Deep Galerkin Network* (DGM) yaitu penyelesaian dengan memanfaatkan *neural network* yang didalamnya terdiri atas kombinasi linear dari beberapa fungsi seperti pada metode penyelesaian *Galerkin* (Sirignano & Spiliopoulos, 2018).

Dalam penelitian ini akan dilakukan pendekatan penghitungan pada persamaan diferensial yang dikembangkan untuk menghitung persamaan diferensial pada kasus difusi. Proses penghitungan pada kasus persamaaan diferensialnya dilakukan dalam 2 bentuk, yaitu penghitungan pada syarat batas dan kondisi awal, kemudian pendekatan kedua yaitu penghitungan persamaan diferensial dengan model *neural network* (Lagaris, Likas, & Fotiadis, 1998). Kerangka kerja yang diterapkan yaitu dengan membuat model *neural network* yang mampu melakukan pendekatan fungsi untuk masalah persamaan diferensial secara *feedforward*, yang mana bagian tersebut merupakan elemen utama untuk melakukan pendekatan yang melibatkan pengaturan parameter bias dan bobot.

1.2 Rumusan Masalah

Dari latar belakang tersebut, maka diperoleh beberapa rumusan masalah sebagai berikut:

- Bagaimana penerapan model *artificial neural network* untuk menyelesaikan persamaan diferensial parsial?
- Bagaimana permasalahan syarat batas persamaan diferensial parsial dapat diselesaikan dengan *neural network*?
- Bagaimana perbandingan perhitungan persamaan diferensial parsial dengan *neural network* jika dibandingkan dengan metode numerik biasa?

.3 Batasan Masalah

Pada Penelitian ini permasalahan dibatasi pada hal-hal berikut:

- penghitungan akurasi prediksi model *neural network* yang digunakan untuk menyelesaikan kasus difusi yang dibandingkan terhadap solusi analitik. Kecepatan penghitungan tidak menjadi fokus pada penelitian.
- Kasus persamaan diferensial yang diselesaikan adalah persamaan diferensial yang bersifat homogen dan permasalahannya didekati dengan *boundary value problem*.

.4 Tujuan

Dari latarbelakang dan rumusan masalah, maka tujuan dari enelitian ini yaitu:

- Mengetahui bagaimana menerapkan model *neural network* dapat menyelsaikan persamaan diferensial parsial;
- Mengetahui cara penentuan syarat batas untuk penyelesaian persamaan diferensial parsial dengan pendekatan *neural network*;
- Mengetahui perbandingan persamaan diferensial parsial yang dihitung secara numerik dengan pendekatan *neural network* dengan metode numerik biasa.

.5 Manfaat

Manfaat dari penelitian ini yaitu:

- Diperoleh suatu metode yang efektif pada penyelesaian persamaan diferensial secara numerk dengan model *neural network*.
- Menemukan solusi untuk mengatasi syarat batas dan penyelesaian persamaan difusi.

BAB II
TINJAUAN PUSTAKA

2.1 Persamaan Diferensial

Persamaan diferensial merupakan bentuk persamaan yang mengandung unsur *derivative* (turunan) (Boyce & DiPrima, 2001). Turunan merupakan perubahan terkecil dari suatu fungsi terhadap varibel tertentu. Perubahan yang sangat kecil ini merupakan delta dengan nilai yang mendekati nol. Biasanya dinyatakan dengan bentuk limit. Missal pada turunan suatu fungsi $U(x)$ yang diturunkan terhadap x dapat dinyatakan dalam bentuk limit yaitu $\frac{dU}{dx} = \lim_{\Delta x \to 0} \frac{U(x+\Delta x)-U(x)}{\Delta x}$

Bentuk persamaan difenrensial dibagi menjadi dua yaitu persamaan diferensial biasa dan persamaan diferensial parsial. Bentuk umum persamaan diferensial biasa pada n-orde dapat dinyatakan dengan :

$$F(x, y(x), y'(x), \dots, y^{(n)}) = 0 \qquad (2.1)$$

Jika diketahui persamaan diferensial biasa pada orde pertama dengan kondisi awal tertentu maka persamaan diferensial tersebut dapat dinyatakan sebagai berikut:

$$\begin{aligned} y'(x) &= f(x, y(x)) \\ y(x_0) &= y_0 \end{aligned} \qquad (2.2)$$

Dalam persamaan (2.2) f merupakan fungsi *real* atas dua variabel x, y dan $x_0\ y_0$ yang bersifat real. Untuk persamaan diferensial biasa orde dapat dituliis dengan persamaan berikut:

$$y''(x) = f(x, y(x), y'(x)) \qquad (2.3)$$

Persamaan (2.3), memiliki solusi umum dengan dua parameter yang berkaitan dengan keadaan awal, yang dapat dinyatakan sebagai berikut:

$$\begin{aligned} y''(x) &= f\big(x, y(x), y'(x)\big) \\ y(x_0) &= y_0, y(x_1) = y_1 \end{aligned} \qquad (2.4)$$

ada persamaan (2.4) variabel y_0 dan y_1 telah diketahui nilainya.

Sebuah persamaan diferensial parsial orde dua untuk suatu ungsi $u(x,y)$ dapat dinyatakan dalam persamaan berikut:

$$F(x,y,u,u_x,u_y,u_{xx},u_{xy},u_{yy}) = 0 \quad (2.5)$$

ecara linear persamaan diferensial parsial orde pertama dapat inyatakan pada persamaan berikut:

$$a(x,y)u_x + b(x,y)u_y + c(x,y)u = f(x,y) \quad (2.6)$$

engan nilai a, b, c dan f diketahui. Persamaan dapat bersifat uasilinear pada persamaan diferensial dengan orde tinggi, misalnya ntuk persamaan diferensial parsial orde kedua (persamaan 2.7), yaitu ebagai berikut:

$$a(x,y,u,u_x,u_y)u_{xx} + b(x,y,u,u_x,u_y)u_{xy} + a(x,y,u,u_x,u_y)u_{yy} = 0 \quad (2.7)$$

Secara umum, persamaan diferensial parsial dapat ikelompokkan menjadi 3 jenis, yaitu persamaan diferensial eliptik, arabolik dan hiperbolik. Conoth dari persamaan eliptik adalah ersamaan Poisson, yang dinyatakan sebagai berikut:

$$\frac{\partial u}{\partial x^2}(x,y) + \frac{\partial u}{\partial y^2}(x,y) = f(x,y) \quad (2.8)$$

ka nilai $f(x,y) = 0$, maka persamaan (2.8) akan menjadi lebih ederhana,

$$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = 0 \quad (2.9)$$

ersamaan (2.9) disebut sebagai persamaan *Laplace*. Dalam ilmu fisika ersamaan *Laplace* dapat digunakan untuk memodelkan kasus difusi anas dalam keadaan tunak pada benda 2 dimensi atau 3 dimensi.

Persamaan diferensial parsial parabolik merupakan persamaan iferensial parsial yang melibatkan variabel waktu. Contoh dari

persamaan ini adalah persamaan difusi dalam fungsi waktu t, yang dinyatakan dalam persamaan berikut.

$$\frac{\partial u}{\partial t}(x,t) + \frac{\partial u}{\partial y^2}(x,t) = 0 \quad (2.10)$$

Kemudian, persamaan yang terakhir adalah persamaan hiperbolik yang melibatkan bentuk turunan parsial orde dua pada variabel t. Contoh dari persamaan ini adalah persamaan gelombang, yang dinyatakan sebagai berikut:

$$\alpha^2 \frac{\partial^2 u}{\partial x^2}(x,t) + \frac{\partial^2 u}{\partial t^2}(x,t) = 0 \quad (2.11)$$

2.2 Contoh Kasus Persamaan Diferensial Parsial

2.2.1 Persamaan *Laplace*

Persamaan *Laplace* merupakan persamaan diferensial parsial orde dua yang bersifat eliptik (Arfken, Weber, & Harris, 2013). Bentuk persamaan *Laplace* dinyatakan sebagai berikut:

$$\nabla^2 \psi(x,y) = 0 \quad (2.12)$$

$$\frac{\partial^2}{\partial x^2}\psi(x,y) + \frac{\partial^2}{\partial y^2}\psi(x,y) = 0 \quad (2.13)$$

Simbol ∇^2 disebut dengan operator *Laplace* yang merupakan bentuk dari operator divergensi dari $\nabla \cdot \nabla$, yang diartikan jika operator tersebut dikenai pada fungsi $f(x,y)$ maka akan didiferensialkan dua kali. Operator *Laplace* melakukan pemetaan fungsi skalar ke fungsi skalar lainnya.

Secara teori, solusi dai persamaan *Laplace* diketahui sebagai teori potensial yang menghasilkan fungsi harmonik. Pada kasus konduksi panas, persamaan *Laplace* dapat diterapkan pada kasus keadaan tunak.

2.2.2 Persamaan Difusi

Pada persamaan difusi satu dimensi akan diterapkan persamaan diferensial parsial orde dua. Persamaan difusi 1 dimensi dinyatakan sebagai berikut :

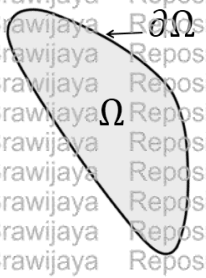
$$\frac{\partial U}{\partial t} = v \frac{\partial^2 U}{\partial x^2} \quad (2.14)$$

Persamaan (2.14) merupakan pernyataan dari Hukum *Fick* yang menyatakan bagaimana konsentrasi difusi berubah terhadap waktu. Konsentrasi suatu zat dinyatakan dalam $U(x,t)$ yang bergantung pada besarnya waktu (t) dan posisi (x). Koefisien difusi dinyatakan dengan v yang merepresentasikan percepatan difusi dan x menyatakan posisi. Pada keadaan dua dimensi dinyatakan dengan *Laplacian* sebagai berikut:

$$\frac{\partial u}{\partial t} = v\nabla^2 U \quad (2.15)$$

.3 Kondisi Awal dan Syarat Batas pada Persamaan Diferensial

Secara umum, untuk menyelesaikan kasus fisis akan ihubungkan keadaan tertentu sistem berdasarkan hukum yang endasari proses fiskal tersebut, hal ini dimaksudkan untuk emudahkan dalam memprediksi rentetan peristiwa. Misalnya, definisi eadaan sistem dapat dinyatakan dalam keadaan awal sistem dan syarat atas. Dengan mengetahui poin-poin tersebut, tentunya akan lebih udah untuk mencari solusi kasus fisis, terutama pada penerapan ersamaan diferensial.



***Gambar 2.1**. Penggambaran domain dan batas dari suatu sistem*

Kondisi awal pada kasus difusi merupakan distribusi temperatur awal pada sistem (Boyce & DiPrima, 2001; Hancock, 2006). Misalnya pada kasus distribusi panas pada suatu batang, kondisi awal dinyatakan dengan $u(x, 0)$. Syarat batas (Gambar 2.1) merupakan konstrain terhadap solusi persamaan diferensial yang diselesaikan pada domain tertentu dimana beberapa kondisinya diketahui. Sedangkan keadaan awal suatu sistem merupakan keadaan ekstrim yang diketahui pada suatu interval tertentu. Penentuan syarat batas merupakan hal yang mendasar dalam proses komputasi, penempatan syarat batas yang salah akan memberikan keadaan yang divergen sehingga solusi menjadi tidak tepat. Berikut dijelaskan beberapa jenis syarat batas yang sering digunakan dalam penyelesaian persamaan diferensial.

2.3.1 Syarat batas *Dirichlet*

Keadaan syarat batas ini spesifik pada nilai dari fungsi yang diketahui nilainya sepanjang batas domain (Arfken et al., 2013). Misalnya pada persamaan *Laplace* dengan syarat batas *Dirichlet* dinyatakan sebagai berikut:

$$\Delta\varphi(x) = 0 \qquad \forall x \in \Omega \tag{2.16}$$

$$\varphi(x) = f(x) \qquad \forall x \in \partial\Omega \tag{2.17}$$

Dimana φ merupakan fungsi yang tidak dikethaui, nilai x merupakan variabel bebas dengan Ω adalah domain dari fungsi, sedangkan $\partial\Omega$ menyatakan batas dari domain, dan f diberkan sebagai fungsi skalar yang dinyatakan pada $\partial\Omega$.

2.3.2 Syarat batas *Neumann*

Syarat batas *Neumann* memberikan spesifikasi nilai batas yang merupakan bentuk turunan pada domain batasnya. Keadaan Neumann seing digunakan untuk menggambarkan sistem yang terinsulasi di salah satu syarat batasnya. Syarat batas *Neumann* dinyatakan sebagai berikut pada persamaan *Laplace*:

$$\Delta\varphi(x) = 0 \qquad \forall x \in \Omega \tag{2.18}$$

$$\frac{\partial\varphi(x)}{\partial n} = f(x) \qquad \forall x \in \partial\Omega \tag{2.19}$$

2.3.3 Syarat batas *Robin*

Syarat batas *Robin* terdiri atas kombinasi nilai dari medan dan turunan dari batasnya. Misalnya pada persamaan *Laplace* syarat batas *Robin* dinyatakan sebagai berikut:

$$\Delta\varphi(x) = 0 \qquad \forall x \in \Omega \qquad (2.20)$$

$$a\varphi(x) + b\frac{\partial\varphi(x)}{\partial n} = f(x) \qquad \forall x \in \partial\Omega \qquad (2.21)$$

Nilai a dan b merupakan real. Keadaan pada sayarat batas *Robin* juga disebut sebagai kondisi impedansi

2.3.4 Syarat batas campuran

Syarat batas campuran terdiri atas beberapa jenis syarat batas yang berbeda pada bagian domain yang berbeda pula. Hal yang perlu diperhatikan dalam mengatur syarat batas campuran yaitu syarat batas harus diaplikasikan pada keseluruhan batas, yang mana batas yang bersifat bebas merujuk pada kondisi *Neumann* yang homogen.

.4 Metode Numerik Penyelesaian Persamaan Diferensial Parsial

Pada beberapa kasus, terdapat beberapa persamaan diferensial ang sulit diselesaikan secara analitik, misalnya pada penyelesaian ersamaan *Navier-Stokes* yang mana solusinya akan lebih mudah idapatkan dengan menggunakan pendekatan analitik. Metode *Finite Difference* merupakan metode numerik yang popular untuk enyelesaikan persamaan diferensial parsial, karena metode ini erupakan metode yang sederhana Penyelesaian persamaan diferensial engan *finite difference* dilakukan dengan melakukan pendekatan perator diferensial secara diskrit (Al-Aradi, Correia, Naiff, & Jardim, 018). Bentuk sederhana dari metode ini adalah metode *Euler*, *Forward ime Centre Space* (FTCS), dan *Backward Time Centre Space* (BTCS).

1. Metode *Euler*

Metode Euler merupakan metode *finite difference* paling sederhana. Pendekatan *Euler* dapat didapatkan dari

ekspansi *deret Taylor* (Giordano & Nakanishi, 2006). Berikut pada persamaan (2.22) menyatakan ekspansi deret *Taylor* fungsi U yang diturunkan terhadap Δx.

$$U(\Delta x) = U(0) + \frac{dU}{dx}\Delta x + \frac{1}{2}\frac{d^2U}{dx^2}(\Delta x)^2 + \cdots \quad (2.22)$$

$U(0)$ merupakan nilai fungsi yang dihitung pada ruang $x = 0$, sedangkan $U(\Delta)$ adalah nilai fungsi pada $x = \Delta x$. Jika nilai Δx dibuat lebih kecil maka akan kita diperoleh hasil pendekatan yang baik. Nilai Δx yang sangat kecil bukan nol dapat dinyatakan dengan pedekatan limit.

Dari persamaan (2.22), dapat diperoleh pendekatan *Euler* untuk orde pertama dengan melakukan pemotogan (*truncation*) pada orde pertama. Misal jika diketahui suatu notasi turunan $\frac{dU}{dx}$, dimana fungsi U diturunkan terhadap variabel x pada domain ruang. Maka untuk pendekatan numerik *Euler*-nya dinyatakan dengan persamaan berikut:

$$\frac{dU}{dx} = \lim_{\Delta x \to 0} \frac{U(x+\Delta x) - U(x)}{\Delta x} \quad (2.23)$$

Atau dapat ditulis sebagai berikut:

$$\frac{dU}{dx} \approx \frac{U(x+\Delta x) - U(x)}{\Delta x} \quad (2.24)$$

Maka pendekatan hasil turunan secara numerik dapat diperoleh sebagai berikut:

$$U(x + \Delta x) \approx U(x) + \frac{dU}{dx}\Delta x \quad (2.25)$$

2. FTCS (*Forwad Time Centre Space*)

Metode FTCS merupakan pengembangan dari metode euler sederhana untuk menyelesaikan persamaan diferensial parsial. Metode ini digunakan untuk menyelesaikan permasalahan dalam dimensi ruang dan waktu, misalnya pada persamaan difusi yang bergantung pada penurunan terhadap ruang dan waktu.

Pada persamaan difusi satu dimensi akan diterapkan persamaan diferensial parsial orde dua. Persamaan difusi 1 dimensi dinyatakan sebagai berikut:

$$\frac{\partial u}{\partial t} = v\frac{\partial^2 u}{\partial x^2} \quad (2.26)$$

Dengan mengacu pada metode *Euler* sebelumnya, maka turunan terhadap t akan dihitung secara *forward difference*, sedangkan untuk turunan terhadap x dinyatakan dengan metode *central difference*.

Bentuk turunan kedua untuk *central difference* pada dimensi ruang dinyatakan sebagai berikut:

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1}-2u_i+u_{i-1}}{\Delta x^2} + O(\Delta x^2) \quad (2.27)$$

Notasi i menunjukkan langkah diskrit dalam domain ruang. Sedangkan untuk bagian domain waktu, dinyatakan secara *forward* sebagai berikut:

$$\frac{\partial u}{\partial t} = \frac{u_i^{n+1}-u_i^n}{\Delta t} \quad (2.28)$$

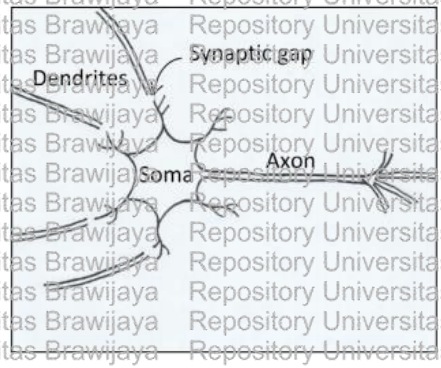
Cacahan waktu dinyatakan pada notasi n dan dicacah secara maju. Dengan menggabungkan persamaan (2.27) dan (2.28), maka diperoleh solusi persamaan difusi sebagai berikut.

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} = v\frac{u_{i+1}^n-2u_i^n+u_{i-1}^n}{\Delta x^2} \quad (2.29)$$

.5 Artificial Neural Network

Artificial neural network (Jaringan saraf buatan) merupakan eknik yang digunakan untuk pemrosesan informasi yang ekanismenya didasarkan pada jaringan neural biologis (gambar 2.2) Zaccone, 2016). Setiap informasi akan diproses melalui tiap *neuron* ang terkoneksi satu sama lain bergantung pada arsitektur yang dibuat. rsitektur *neural network* akan mendeskripsikan koneksi antar *neuron*, umlah lapisan, dan jumlah *neuron* ditiap lapisan. koneksi tersebut bisa

berupa *feedforward neural network*, *recurrent*, *multi* atau *single layer neural network*.

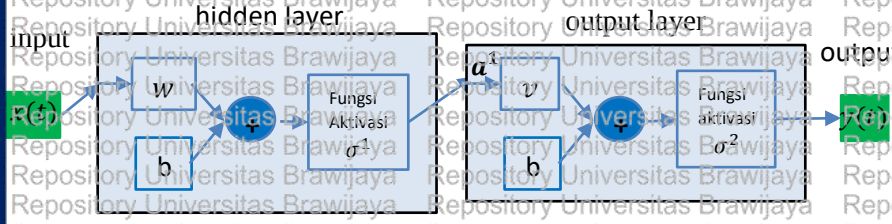


Gambar2.2 Neuron pada saraf biologi berfungsi untuk mentransfer informasi dari input alat indera, sedangkan pada ANN *neuron* akan menerima nilai input didasarkan pada parameter yang telah didefinisikan.

Gambar 2.3. Skema proses antara *neuron* biologi dan ANN

Karakteristik lain dari *neural network* selain arsitekturnya yaitu Proses *learning* (pembelajaran). Proses ini menjelaskan tentang training proses dari algoritma yang dibangun. Kerakteristik berikutnya adalah fungsi aktivasi yang menjelaskan tentang nilai aktivasi yang dilewatkan

ada tiap neuron. Fungsi aktivasi menjelaskan juga bagaimana *neuron* ekerja misalnya secara stokastik atau linear. Secara proses, arsitektur NN dapat direpresntasikan pada diagram gambar 2.4.



Gambar 2.4.Gambar arsitektur *neural network* sederhana pada satu *hidden layer*, indeks atas pada tiap notasi a^1, σ^1 menunjukkan urutan layer ke 1 dan seterusnya. Model ini juga disebut sebagai *feedforward neural network*.

Sel biologi terdiri atas dendrit, soma (badan sel), akson, dan naptik (Gambar 2.2). Bagian-bagian tersebut memiliki analogi fungsi engan ANN. Dendrit akan membawa input dari *neuron* ke soma (badan el). Kemudian pada soma, input akan diproses dan dijumlahkan. emudian input akan ditransmisikan ke *neuron* lain melalui akson. aka secara analogi proses (Gambar 2.3) pada ANN dendrit merupakan oneksi input, soma adalah bagian aktivasi/fungsi aktivasi, akson erupakan koneksi *output*, dan sinaptik adalah bobot pada ANN.

2.5.1 Model Artificial Neural Network

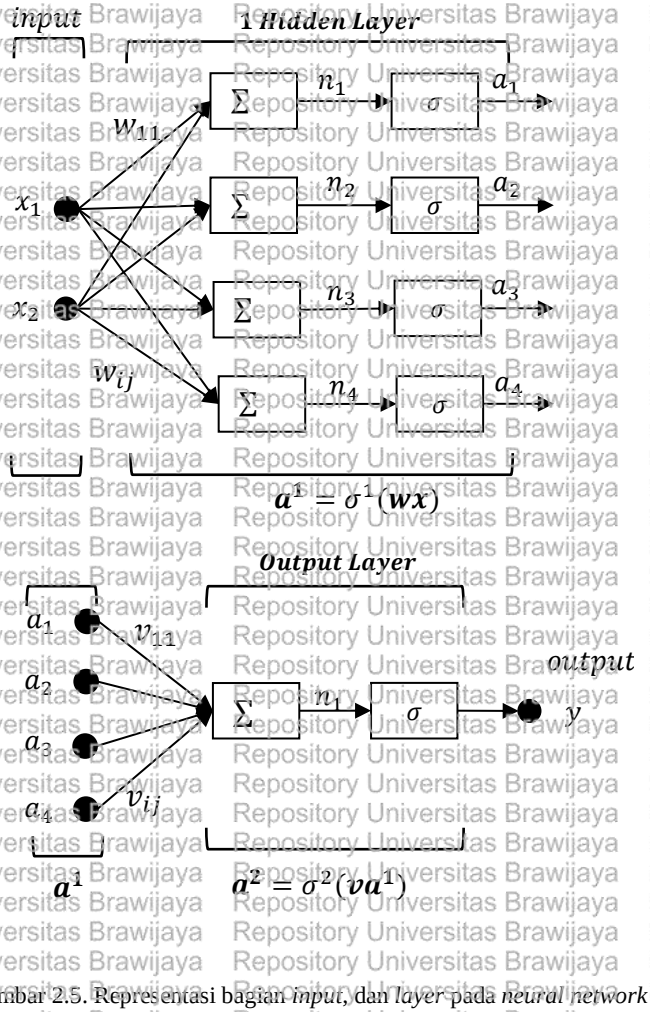
Jika diketahui suatu neural network yang terdiri atas n input dengan total satu *hidden layer* dengan input dalam bentuk vektor berukuran $R \times 1$ dinyatakan dengan $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_R)$ seperti pada Gambar 2.5 dengan fungsi aktivasi sigmoid σ^i yang diindekskan sesuai dengan urutan layer ke i dan outputnya bersifat *linear* maka output *neural network* dapat dinyatakan sebagai berikut :

$$output = \sigma^2(\boldsymbol{va^1}) \quad (2.29)$$

Dengan nilai $\boldsymbol{a^1}$ adalah hasil dari perhitungan pada tiap neuronnya di layer 1 (urutan layer ditujukan dengan indeks i pada notasi $\boldsymbol{a^i}$) yang dinyatakan pada persamaan (2.30).

$$\boldsymbol{a^1} = \sigma^1(\boldsymbol{wx}) \tag{2.30}$$

parameter bobot pada *hidden layer* dinyatakan dengan w sedangkan untuk parameter bobot output layer dinayatakan dengan v dan untuk parameter bias dinyatakan dengan b.



Gambar 2.5. Representasi bagian *input*, dan *layer* pada *neural network*

Misal diberikan suatu arsitektur *neural network* yang terdiri atas 1 *hidden layer* dengan 4 *neuron* dan 2 *input* dengan *output* berjumlah 1 seperti pada Gambar 2.5, maka penghitungan *neural network* akan dilakukan sebagai berikut.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\boldsymbol{w} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix}$$

$$\boldsymbol{n} = \mathbf{x} \cdot \boldsymbol{w} = \begin{bmatrix} x_1 w_{11} + x_2 w_{21} \\ x_1 w_{21} + x_2 w_{22} \\ x_1 w_{31} + x_2 w_{32} \\ x_1 w_{41} + x_2 w_{42} \end{bmatrix} = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_5 \end{bmatrix}$$

$$\boldsymbol{a^1} = \sigma^1(\boldsymbol{n}) = \begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \\ a_4^1 \end{bmatrix}$$

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}$$

$$output = \sigma^2(\boldsymbol{a^1} \cdot \boldsymbol{v}) = \sigma^2(a_1^1 v_1 + a_2^1 v_2 + a_3^1 v_3 + a_4^1 v_4)$$

2.5.2 *Feed Forward Neural Network*

Feedforward neural network atau dapat disebut dengan multilayer perceptron merupakan model *neural network* yang klasik atau sederhana (Gambar 2.4). Model ini disebut *feedforward* karena informasi akan diproses melalui fungsi yang dievaluasi dari x secara maju dihasilkan suatu ouput tanpa ada koneksi *feedback* dalam prosesnya (Goodfellow et al., 2016).

Model ini dibuat untuk melakukan pendektan terhadap suatu fungsi f. Misalnya, pada kasus *classifier*, $y = f*(x)$ memetakan input x ke sebuah kategori y. Arsitektur *feedforward network* akan melakukan pemetaan $y = f(x; \theta)$ dan mempelajari tiap besar parameter θ yang akan menghasilkan pendekatan yang terbaik terhadap fungsi.

Feedforward neural network secara ciri khususnya menampilkan proses network yang menggabungkan beberapa fungsi yang berbeda secara bersamaan. Sebagai contoh, jika diketahui suatu fungsi $f^{(1)}, f^{(2)}, f^{(3)}$ yang dikoneksikan secara berantai membentuk $f(x) = f^{(1)}(f^{(2)}(f^{(3)}))$, yang mana struktur rantai tersebut sangat umum digunakan pada *neural network*, dimana $f^{(1)}$ merupakan *layer* pertama pada *neural network*, $f^{(2)}$ disebut sebagai layer ke 2 dan seterusnya.

2.5.3 *Stochastic Gradient Descent* (SGD)

Stochastic gradient descent (SGD) adalah algoritma yang diterapkan untuk *tuning* parameter bias dan bobot. Metode ini disebut stokastik karena *sample* dari parameter diambil secara acak pada proses *training* (Sena, 2017). Perhitungan parameter dengan *gradient* dinyatakan dengan persamaan berikut:

$$w_{k+1} := w_k - \alpha \left(\frac{\partial Loss}{\partial w_k}\right) \tag{2.31}$$

Parameter bobot w akan diperkirakan dengan melakukan iterasi pada *epoch* sejumlah $k+1$ hingga diperoleh nilai bobot yang optimum w_{k+1} pada *learning step* sebesar α. Untuk membuat proses penghitungan lebih signifikan, maka pada proses training akan dibagi menjadi beberapa *batch* (membagi proses taaing menjadi beberapa bagian).

Sebagai contoh dilakukan pendekatan suatu garis lurus $y = \sigma(w_1 x_1 + w_2 x_2)$ pada data training yang telah ditentukan $(x_1, x_2, x_3, \ldots, x_n)$ dengan target data $(\hat{y}_1, \hat{y}_2, \hat{y}_3, \ldots, \hat{y}_n)$ diestimasi dengan metode *least square*.

$$Loss(w) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$
$$Loss(w) = \sum_{i=1}^{n} (\sigma(w_1 x_1 + w_2 x_2) - y_i)^2 \tag{2.32}$$

Maka parameter akan di-*update* sebagai berikut:

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}^{k+1} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}^{k} - \alpha \begin{bmatrix} \frac{\partial}{\partial w_1}(\sigma(w_1x_1 + w_2x_2) - y_i)^2 \\ \frac{\partial}{\partial w_2}(\sigma(w_1x_1 + w_2x_2) - y_i)^2 \end{bmatrix}$$

$$= \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}^{k} - \alpha \begin{bmatrix} 2x_1(\sigma(w_1x_1 + w_2x_2) - y_i)\sigma'_{w_1}(w_1x_1 + w_2x_2) \\ 2x_2(\sigma(w_1x_1 + w_2x_2) - y_i)\sigma'_{w_2}(w_1x_1 + w_2x_2) \end{bmatrix} \quad (2.33)$$

2.5.4 Supervised Learning

Pada supervised learning diberikan input variabel (x) dan output variabel (Y) serta algortima pemetaan yang dinyatakan dalam suatu fungsi $Y = f(X)$. Tujuan dari pendekatan pemetaan fungsi tersebut akan bekerja dengan baik jika dimiliki input data baru (x) sehingga dapat dilakukan prediksi terhadap output variabel (Y). Proses algoritma tersebut akan berhenti saat mencapai tingkat performance yang diterima yang dihitung dari *loss function*.

Sedangkan pada algoritma unsupervised learning, data yang diberikan hanya data input (X) tanpa variabel output yang sesuai. Tujuan dari algoritma ini adalah mempelajari struktur atau distribusi data untuk mempelajari lebih dalam tentang data tersebut.

.6 Metrik Evaluasi

Metrik evaluasi atau metrik error digunakan untuk mengetahui agaimana performa suatu model. Maka dari mengetahui nilai metrik valuasi dapat diketahui keakurasian model yang diperoleh. Dengan ata lain sebelum dilakukan testing dari model perlu dicek bagaimana eadaan metrik errornya. Pada kasus machine learning, jenis metrik valuasi yang sering diterapkan adalah Metrik regresi seperti: *Mean quared Error* (MSE), *Root Mean Squarred Error* (RMSE), *Sum quared Error* (SSE).

1. ***Mean Squared Error* (MSE)**

Bentuk dari MSE dinyatakan pada persamaan berikut:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad (2.34)$$

y_i = data validasi

$\hat{y}_i$ = data hasil prediksi model

N = jumlah banyaknya data

Secara mendasar, MSE menghitung nilai kuadrat dari *error* prediksi. Pada tiap data akan dihitung kuadrat perbedaan antar hasil prediksi dengan data validasi kemudian dirata-rata. Semakin besar nilai MSE maka semakin buruk hasil prediksi yang diperoleh. Untuk menerapkan metrik ini perlu diperhatikan pada penerapan data yang nilainya sangat besar atau sangat rendah.

2. ***Root Mean Squarred Error* (RMSE)**

Bentuk RMSE merupakan MSE yang diakarkan. Peskoran nilai RMSE sama dengan nilai MSE, yaitu semakin tinggi nilainya maka prediksi yang dihasilkan cukup buruk. Bentuk dari RMSE dinyatakan pada persamaan berikut:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \quad (2.35)$$

y_i = data validasi

$\hat{y}_i$ = data hasil prediksi model

N = jumlah banyaknya data

3. ***Sum Squared Error* (SSE)**

SSE mengukur variansi dari data yang diukur. Bentuk dari SSE dinyatakan pada persamaan berikut:

$$SSE = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad (2.36)$$

y_i = data validasi

$\hat{y}_i$ = data hasil prediksi model

N = jumlah banyaknya data

BAB III
METODE PENELITIAN

3.1 Waktu dan Tempat

Penelitian ini telah dilaksanakan pada bulan Februari 2019 hingga April 2019 di Laboratorium Simulasi dan Pemodelan, Gedung Biomol, Lantai.3, Jurusan Fisika Universitas Brawijaya.

3.2 Peralatan dan Bahan

Alat yang digunakan dalam penenlitian ini antara lain: komputer dengan spesifikasi RAM minimal 4 GB dan *hardisk* minimal 100 GB yang telah terinstall *Python* versi 2.7 dan modul komputasi *Matplotlib*, *Autograd*, *Scikit learn*, dan *Numpy*.

3.3 Prosedur Penelitian

Pada penelitian ini terdiri dari beberapa tahapan yang dilakukan, yaitu sebagai berikut:

1. Persiapan komputasi, meliputi instalasi *software*.
2. Pembuatan model solusi umum persamaan diferensial dan penentuan syarat batas dalam bentuk kandidat solusi *neural network*.
3. Pra-pemrosesan data.
4. *Proses Training*.
5. *Proses Testing*/Pengujian.

Bagan kerja komputasi dijelaskan pada gambar 3.1.

3.3.1 Persiapan komputasi

Pada tahap ini dilakukan instalasi beberapa *software* yang digunakan dalam komputasi yaitu bahasa pemrograman *Python*, untuk itu dilakukan instalasi *interpreter Python* dengan *environment* 2.7. *Module* yang dibutuhkan untuk melakukan komputasi ini antara lain: *Matplotlib* dan *Autograd* Penggunaan module *Autograd* bertujuan untuk kemudahan pengnitungan

matriks *Jacobian* dan *Hessian* pada komputasi *neural network* (Honchar, 2017). Cara instalasi *Autograd* pada *Python* adalah dengan menggunakan perintah *'pip install autograd'* yang ditulis pada *command prompt* (Dougal Maclaurin, David Duvenaud, Matt Johnson, 2018). Untuk menambahkan *module Matplotlib*, maka digunakan perintah instalasi pada *prompt 'pip install matplotlib'*. Fungsi dari penambahan modul *matplotlib* adalah untuk menampilkan grafik data perhitungan dan visualisasi. Kemudian *Scikitlearn* ditambahkan untuk membantu dalam *pre-processing* (pra-pemrosesan) dan penghitungan nilai ralat antara solusi analitik, numerik, dan solusi dari model kandidat *neural network*. Cara instalasi *module scikitlearn* adalah dengan menuliskan perintah *'pip install scikit-learn'* pada *command prompt*.

3.3.2 Pembuatan Model Solusi Umum

Model solusi umum diadaptasi dari penelitian sebelumnya dengan membagi dua bagian arsitektur *neural network* dan bagian syarat batas (Lagaris et al., 1998). Model solusi untuk persamaan difusi yang tidak bergantung pada waktu diselesaikan secara umum dengan persamaan *Laplace* sebagai berikut:

$$\frac{\partial^2}{\partial x^2}\Psi(x,y)+\frac{\partial^2}{\partial y^2}\Psi(x,y)=f(x,y) \tag{3.1}$$

Dimana $x \in [0,1]$ dan $y \in [0,1]$.

1. Kasus dengan Syarat Batas Dirichlet

Jika persamaan diferensial dengan syarat batas *Dirichlet* sebagai berikut: $\Psi(0,y)=f_0(y)$, $\Psi(1,y)=f_1(y)$ dan $\Psi(x,0)=g_0(x)$, $\Psi(x,1)=g_1(x)$, maka solusi dari *neural network* dari Lagaris,dkk (1998):

$$\Psi_t(x,y)=A(x,y)+x(1-x)y(1-y)N(x,y,\vec{p}) \tag{3.2}$$

$A(x,y)$ merupakan bagian yang digunakan untuk memenuhi keadaan syarat batas model, dinyatakan dengan:

$$A(x,y)=(1-x)f_0(y)+xf_1(y)+(1-y)\{g_0(x)-[(1-x)g_0(0)+xg_0(1)]\}+y(g_1(x)-[(1-x)g_1(0)+xg_1(1)]\} \tag{3.3}$$

Bentuk $N(x,y,\vec{p})$ merupakan *feedforward neural network* dengan input x dan y beserta parameter bobot yang dinyatakan pada p.

2. Kasus syarat batas Campuran

Jika diketahui kasus dengan suatu syarat batas campuran dengan bentuk sebagai berikut: $\Psi(0,y) = f_0(y)$, $\Psi(1,y) = f_1(y)$ dan $\Psi(x,0) = g_0(x)$, $\frac{\partial}{\partial y}\Psi(x,1) = g_1(x)$, maka solusi neural networknya dinyatakan sebagai berikut:

$$\Psi_t(x,y) = B(x,y) + x(1-x)y\left[N(x,y,\vec{p}) - N(x,1,\vec{p}) - \frac{\partial}{\partial y}N(x,1,\vec{p})\right] \quad (3.4)$$

Dan $B(x,y)$ menyatakan syarat batas dari persamaan diferensial yang memenuhi bentuk berikut:

$$B(x,y) = (1-x)f_0(y) + xf_1(y) + g_0(x) - [(1-x)g_0(0) + xg_0(1)] + y\{g_1(x) - [(1-x)g_1(0) + xg_1(1)]\} \quad (3.5)$$

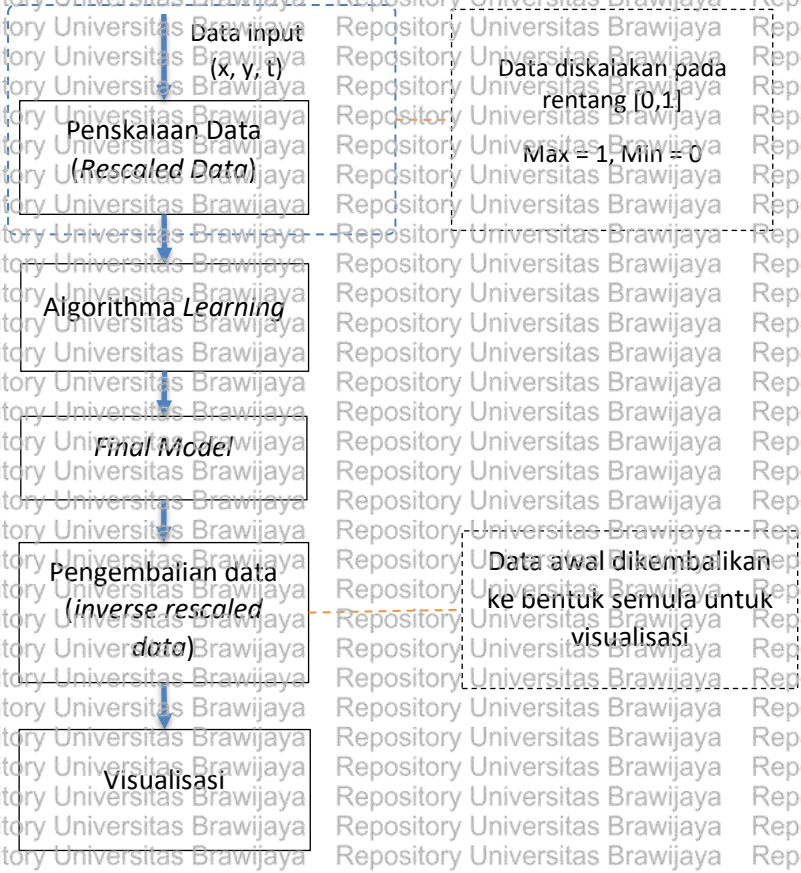
3.3.3 *Pre-processing* Data

Sebelum proses training dilakukan, data input dari variabel $x, y,$ dan t akan diskalakan terlebih dahulu agar proses training menjadi optimum. Kualitas dari data akan mempengaruhi proses learning, sehingga data yang tidak bagus dapat menyebabkan proses learning menjadi tidak baik. Tahapan pra-pemrosesan bertujuan untuk: menghindari nilai *null*, standarisasi data, mengatasi kategori variabel tertentu, dan kasus multikolinearitas.

Pada penelitian ini, data akan diolah dari pra-pemrosesan hingga didapatkan model kandidat final kemudian divisualisasikan (Gambar 3.1). *Raw data* atau data *input* pertama akan diproses terlebih dahulu dengan cara diskalakan pada rentang nilai maksimum yaitu 1 dan nilai minimum 0. Kemudian data akan digunakan untuk proses *training*. Pada tahapan berikutnya, jika sudah diperoleh kandidat model *final*, maka data awal yang sudah diskalakan akan dikembalikan ke bentuk semula (*inverse rescale*),

kemudian dengan data awal tersebut diterapkan pada model kandidat untuk divisualisasikan dan dianalisa.

Pada tiap pemrosesan data akan ditemui beberapa kasus yang berbeda bergantung pada algoritma *learning* yang diterapkan. Beberapa algoritma *learning* ada yang tidak memerlukan tahapan pra-pemrosesan data. Misalnya pada kasus persamaan *Laplace* di persamaan (4.9), domain variabel x dan y berada pada rentang [0,1]. Akan tetapi, untuk data dengan rentang maksimum lebih dari satu maka perlu dilakukan penskalaan data, misalnya pada kasus 4.1.2, 4.1.3 dan 4.1.4.



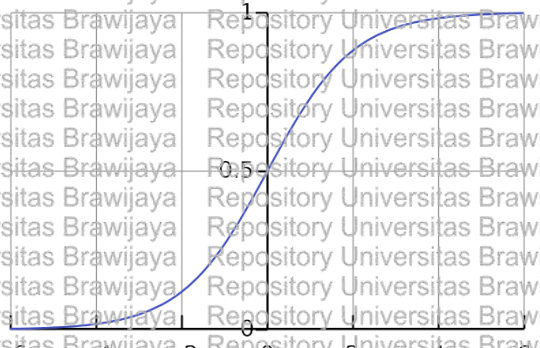
Gambar 3.1. *Data flow* dalam proses *training*

3.3.4 Training *Neural Network*

Untuk melakukan pendekatan suatu fungsi dengan menggunakan *neural network*, pada penelitian ini digunakan teknik *feedforward neural network* dengan satu *hidden layer* yang berisi 10 *neuron*. Fungsi aktivasi yang digunakan adalah sigmoid yang dinyatakan sebagai berikut:

$$\phi(x) = \frac{1}{1+e^{-x}} \tag{3.6}$$

Fungsi aktivasi sigmoid dipilih dalam proses training pada kasus ini karena rentang nilainya berada diantara 0 dan 1, hal ini dikaitkan pula dengan nilai input data training yang rentangnya hanya pada [0,1]. Untuk input yang nilainya lebih besar dari 1, maka akan dilakukan *rescaling* di rentang [0,1] pada tahap *pre-processing* hal ini dilakukan agar proses komputasi tidak pada dearah saturasi.

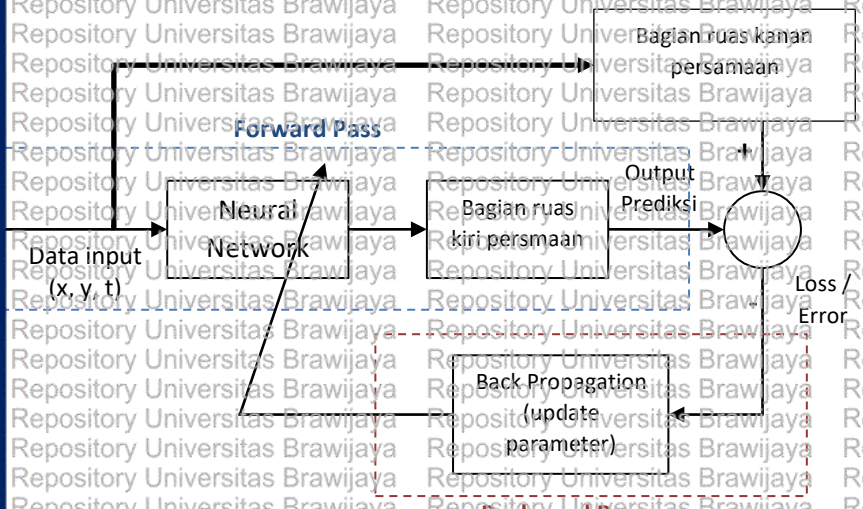


Gambar 3.2. Grafik fungsi aktivasi sigmoid (merah) dan tanh (hijau)

Nilai bobot didapatkan dari proses optimasi dengan metode *gradient descent*. Data *training* diperoleh dari nilai variabel x, y dan t serta nilai syarat batasnya. dari fungsi $\Psi(x, y, t)$.

Proses *forward pass* (Gambar 3.3) melibatkan dua bagian suku persamaan, yaitu bagian suku di sisi kanan dan suku kiri persamaan. Misalnya, untuk kasus persamaan *Poisson*, dari persamaan (3.1) diperoleh suku kanan persamaan yaitu

$\frac{\partial^2}{\partial x^2}\Psi(x,y) + \frac{\partial^2}{\partial y^2}\Psi(x,y)$ dan suku kiri persamaan yang merupakan fungsi $f(x,y)$. Untuk penyelesaian persamaan *Poisson* dan *Laplace* ruas kiri persamaan akan di-*training* bersama dengan *neural network*, sedangkan bagian ruas kanan akan dihitung secara manual diluar *neural network*. Fungsi $f(x,y)$ pada ruas kanan persamaan merupakan bagian pengontrol. Hasil *output* dari proses *forward pass* akan dibandingkan dengan nilai $f(x,y)$, sehingga diperoleh nilai *error/loss* yang akan mengontrol proses training.



Gambar 3.3. Bagan kerja *Neural Network* pada proses training, input data merupakan koordinat ruang (x, y) dan waktu (t),dengan data target fungsi ruas kanan dari persamaan diferensial.

Pada proses training *neural network* ini berbeda dengan *neural network* pada kasus umumnya, yang mana pada kandidat model untuk penyelesaian persamaan diferensial parsial, penghitungan nilai *loss/error* didasarkan pada minimisasi di kedua ruas persamaan hingga didapatkan nilai *loss* yang konvergen, dimana diperoleh nilai *error/loss* mendekati nol atau pada keadaan

perubahan nilai *loss* dari tiap *epoch* tidak menunjukkan perubahan yang signifikan.

Perbedaan berikutnya yaitu pada kandidat model ini, selain pada penentuan *loss function*, data yang digunakan dalam proses training hanya terbatas pada nilai variabel x, y, atau t dan nilai syarat batas. Kemudian data target dari model kandidat diperoleh dari fungsi ruas kanan $f(x, y)$. Dengan kata lain, pada kasus penyelesaian persamaan diferensial, data target seolah-olah bersifat semu karena ditentukan oleh fungsi pengontrol di ruas kanan, sehingga pada bagian pra-pemrosesan tidak dilakukan partisi antara data training, data target, dan data validasi.

Secara rinci, proses training terdiri atas 2 bagian utama yaitu bagian *Forward Pass* dan *Backward Pass*. Dalam *supervised learning*, data input (x, y, t) dipropagasi menuju *layer output* (*Forward-propagation*), kemudian diperoleh hasil prediksi dari neural network dan penghitungan persamaan di ruas kiri, kemudian dibandingkan dengan bagian persamaan di ruas kanan yang dihitung dengan fungsi *loss* (gambar 3.1). Misal, diketahui *multilayer perceptron* dengan input sejumlah n, dengan satu hidden layer dan fungsi aktivasi sigmoid σ, maka output yang diperoleh yaitu $N = \sum_{i=1}^{H} w_i \sigma(z_i)$, dengan nilai $z_i = \sum_{j=1}^{N} w_{ij} x_j + b_i$. Bobot dinyatakan dengan w dan bias b. Dari perhitungan *neural network*, akan diteruskan menuju penghitungan yang didasarkan dari ruas kiri persamaan. *Output* yang diperoleh dari proses *forward pass* akan dibandingkan dengan hasil persamaan di ruas kanan.

Penghiutngan *loss function* akan didekati dengan mencapai nilai konvergen selisih nilai persamaan ruas kanan dengan ruas kiri dengan menggunakan teknik SSE (*Sum Squared Error*). Nilai *error* dihitung dengan menghitung perbedaan persamaan ruas kanan dengan ruas ruas kiri hingga dicapai nilai yang mendekati nol (konvergen). *Loss function* untuk *neural network* penyelesaian persamaan diferensial biasa dinayatakan pada persamaan (3.6), sedangkan untuk penyelesaian persamaan diferensial parsial dinyatakan pada persamaan (3.7).

$$Loss[p] = \sum_{k=1}^{K} \sum_i \left\{ \frac{d\Psi_{t_K}(x_i)}{dx} - f_k(x_i, \Psi_{t_K}) \right\}^2 \quad (3.6)$$

$$Loss[p] = \sum_i \left\{ \frac{\partial^2}{\partial x^2} \Psi(x_i, y_i) + \frac{\partial^2}{\partial y^2} \Psi(x_i, y_i) - f_k(x_i, y_i) \right\}^2 \quad (3.7)$$

Back Propagasi merupakan proses peyesuaian kembali tiap nilai bobot dan bias berdasarkan *error* yang dihasilakn pada proses *forward-propagation*. Tahapan dari *back-propagation* ialah sebagai berikut:

1. Penghitungan gradient dari *loss function* terhadap semua parameter dengan mencari nilai diferensiasi parsialnya.

$$\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial o_{out}} \frac{\partial o_{out}}{\partial o_{in}} \frac{\partial o_{in}}{\partial w_{ij}} \quad (3.7)$$

2. Pembaharuan nilai semua parameter bobot dan bias dengan *stochastic gradient descent* (SGD) dengan mengurangi nilai weight lama dengan learning dari gradient yang sudah diperoleh sebelumnya. Sebagian dari pengurangannya diwakili oleh *hyper-parameter learning rate* (*alpha*).

$$w'_{ij} = w_{ij} - \alpha \left(\frac{\partial Loss}{\partial w_{ij}} \right) \quad (3.8)$$

3.3.5 Testing

Pengujian dilakukan dengan membandingan hasil dari komputasi neural network dengan hasil numerik konvensional dan solusi analitik. Metode numerik yang digunakan untuk menghitung persamaan diferensial adalah FTCS (*Forward Time Center Space*). Perbandingan antar model dihitung dengan RMSE (*Root Mean Square Error*). Selain itu, dari perbandingan antar model diamati juga secara visual dari hasil grafik interpolasi dan membanding besar *error* untuk meyatakan akurasi antara keduanya terhadap solusi eksak.

Repository Universitas Brawijaya

BAB IV
Hasil dan Pembahasan

4.1 Model Neural Network dan Penentuan Syarat Batas

Model *neural network* yang diterapkan untuk mencari solusi persamaan diferensial parsial dibagi menjadi 2 model kandidat yang didasarkan dari syarat batasnya, model yang diterapkan adalah model dari Lagaris, dkk (1998). Pada penelitian ini syarat batas yang digunakan adalah syarat batas *Dirichlet* dan syarat batas *Neumann*.

Pada persamaan *Poisson* dimodelkan sebagai berikut:

$$\frac{\partial^2}{\partial x^2}\Psi(x,y)+\frac{\partial^2}{\partial y^2}\Psi(x,y)=f(x,y) \tag{4.1}$$

Untuk penyelesaian kasus persamaan *Poisson* pada syarat batas *Dirichlet* dinyatakan pada persamaan(3.2), dengan syarat batas sebagai berikut:

$$\begin{aligned}\Psi(0,y)&=f_0(y)\\ \Psi(1,y)&=f_1(y)\\ \Psi(x,0)&=g_0(x)\\ \Psi(x,1)&=g_1(x)\end{aligned} \tag{4.2}$$

Kemudian model kandidat untuk syarat batas dinyatakan dengan persamaan berikut.

$$A(x,y)=(1-x)f_0(y)+xf_1(y)+(1-y)\{g_0(x)-[(1-x)g_0(0)+xg_0(1)]\}+y\{g_1(x)-[(1-x)g_1(0)+xg_1(1)]\} \tag{4.3}$$

Untuk menerapkannya pada kasus fisis, maka nilai dari tiap syarat batas yang diketahui disubtitusikan pada pesamaan (4.3). Kemudian solusi *neural network* akan dinyatakan sebagai berikut:

$$\Psi_t(x,y)=A(x,y)+x(1-x)y(1-y)N(x,y,\vec{p}) \tag{4.4}$$

Nilai $N(x,y,\vec{p})$ menyatakan *neural network* dari input variabel x dan y. Jika diberikan nilai input berupa matriks 2 dimensi yang memuat input variabel x dan y, dinyatakan dalam persamaan (4.5).

$$\vec{x} = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ \vdots & \vdots \\ x_{n,1} & x_{m,n} \end{pmatrix} \quad (4.5)$$

Kemudian hasil keluaran dari tiap layer yang dinyatakan dengan z_i merupakan jumlah dari penghitungan nilai input yang diproses pada sejumlah N neuron yang dinyatakan dengan:

$$z_i = \sum_{j=1}^{N} w_{ij} x_j + b_i \quad (4.6)$$

Nilai parameter $\vec{p}$ yang terdiri atas: $w_{i,j}$ nilai bobot dan b_i nilai bias, yang mana nilai parameter tersebut akan diperbarui di proses training untuk mendapatkan parameter training terbaik. Kemudian, output dari *neural network* berupa output linear. Hasil dai penghitungan pada tiap layer H akan diaktivasi dengan fungsi aktivasi σ *sigmoid* dan dikalikan dengan bobot pada layer aktivasi v_i, maka dari hasil penghitungan akan diperoleh output sebagai berikut:

$$output = \sum_{i=1}^{H} v_i \sigma(z_i) \quad (4.7)$$

Pada kasus syarat batas *Neumann*, yang mana keadaan syarat batas dinyatakan sebagai berikut:

$$\begin{aligned} \Psi(0, y) &= f_0(y) \\ \Psi(1, y) &= f_1(y) \\ \Psi(x, 0) &= g_0(x) \\ \frac{\delta}{\delta y}\Psi(x, 1) &= g_1(x) \end{aligned} \quad (4.8)$$

Maka solusi kandidat dinyatakan dengan:

$$\Psi_t(x, y) = B(x, y) + x(1 - x)y\left[N(x, y, \vec{p}) - N(x, 1, \vec{p}) - \frac{\partial}{\partial y} N(x, 1, \vec{p})\right] \quad (4.9)$$

Kemudian syarat batas diatur pada $B(x, y)$:

$$B(x, y) = (1 - x)f_0(y) + xf_1(y) + g_0(x) - (1 - x)g_0(0) + xg_0(1)] + y\{g_1(x) - [(1 - x)g_1(0) + xg_1(1)]\} \quad (4.10)$$

Penghitungan nilai *error* pada *neural network* untuk kedua kasus syarat batas dilakukan dengan cara yang sama yaitu dengan mencari nilai yang konvergen dari selisih persamaan antara ruas kiri dan ruas kanan dari persamaan (4.1). Maka *loss function* akan diminimalisasi dapat dinyatakan pada persamaan berikut:

$$Loss[p] = \sum_i \left\{ \frac{\partial^2}{\partial x^2}\Psi(x_i, y_i) + \frac{\partial^2}{\partial y^2}\Psi(x_i, y_i) - f_k(x_i, y_i) \right\}^2 \quad (4.11)$$

Dimana dalam persamaan (4.11) nilai x_i, y_i secara berturut-turut berada di rentang $[1,0], [1,0]$.

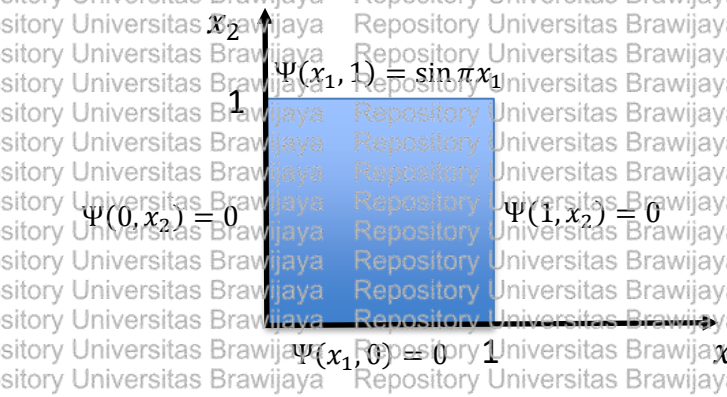
4.1.1 Persamaan *Laplace* (Syarat Batas *Dirichlet*)

Pada kasus pertama diselesaikan masalah pada persamaan *Laplace* Untuk kasus ini diambil dari contoh penelitian sebelumnya oleh Chiaramonte dan Kiener (2013). Masalah ini disketsakan pada Gambar 4.1 dan persamaan model tersebut dinyatakan sebagai berikut:

$$\frac{\partial^2}{\partial x^2}\Psi(x, y) + \frac{\partial^2}{\partial y^2}\Psi(x, y) = 0 \quad (4.12)$$

Dengan syarat batas *Dirichlet*:

$$\begin{aligned} \Psi(x) &= 0 \quad \forall x \in \{(x_1, x_2) \in \partial D \mid x_1 = 0, x_1 = 2, x_2 = 1\} \\ \Psi(x) &= \sin \pi x_1, \quad \forall x \in \{(x_1, x_2) \in \partial D \mid x_2 = 1\} \end{aligned} \quad (4.13)$$



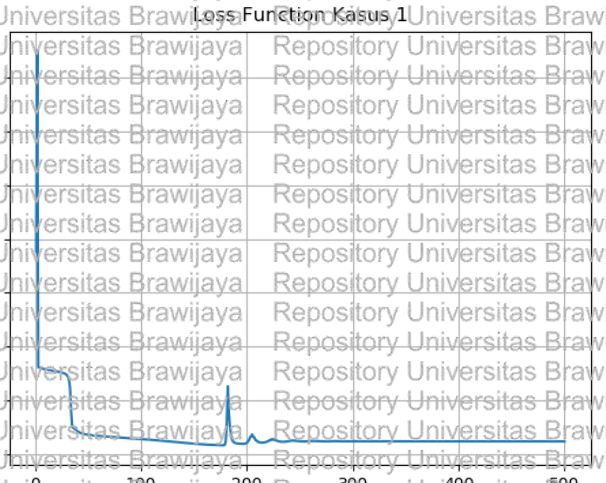
Gambar 4.1. Sketsa kasus persamaan (4.9) dengan keadaan syarat batas pada persamaan (4.10)

Maka solusi kandidat *neural network* yang digunakan adalah persamaan (4.3). Untuk nilai $A(x,y)$ yang memenuhi syarat batas yaitu.

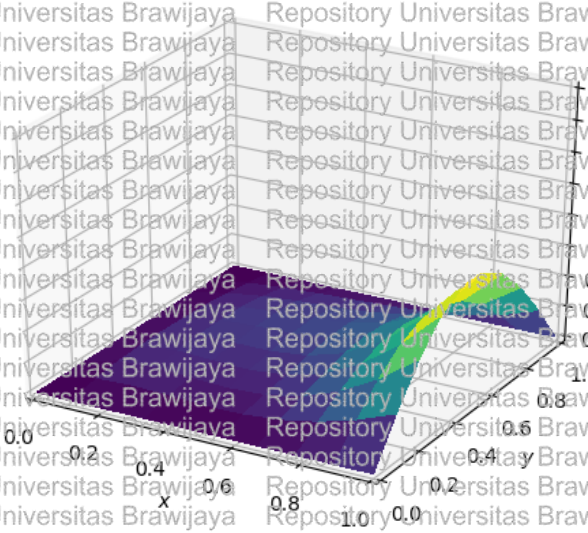
$$A(x,y) = x_1 + 1 + x_2 \sin \pi x \qquad (4.14)$$

Hasil interpolasi yang diperoleh ditunjukkan pada Gambar 4.3. Pada proses training, solusi kandidat *neural network* berhasil mendekati nilai analitik pada *epoch* ke 100 .

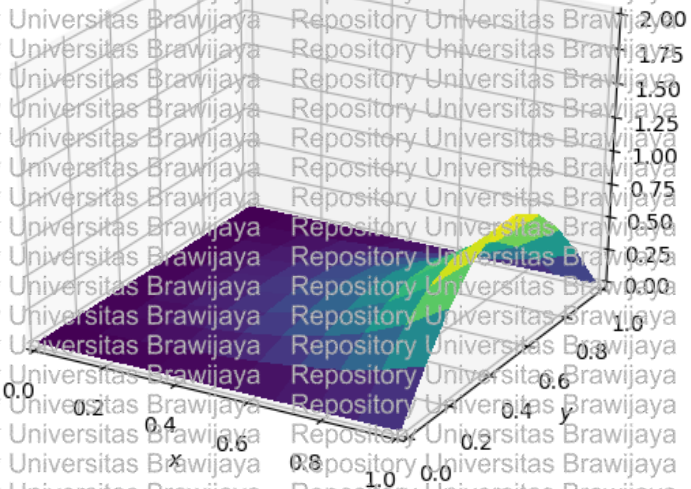
Untuk menyelesaikan persamaan tersebut parameter yang digunakan untuk proses training hanya dengan mengpdate bobot. Hal tersebut dilakukan untuk mendakaui keakuratan nilai. Pada penelitian ini parameter bias dibuat nol, sehingga hanya dilakukan training parameter bobot. *Starting point* nilai bobot ditentukan dengan mengatur nilai *random seed*. Penentuan nilai *random seed* dilakukan secara berulang hingga diperoleh nilai *random seed* yang menghasilkan nilai bobot opimum untuk *training*. Pada pra-pemrosesan, dilakukan uji coba nilai *random seed* dari rentang 1 hingga 50, dan nilai random seed yang optimum adalah 1.



Gambar 4.2. Uji Coba proses training dengan epoch 500, sumbu x menyatakan epoch, dan sumbu y merupakan nilai loss



Gambar 4.3. Hasil Prediksi menggunakan neural network



Gambar 4.4. Hasil perhitungan secara analitik

Kemudian pada proses training dengan 500 *epoch*, didapatkan profil nilai *loss function* (Gambar 4.2) mengalami saturasi pada *epoch* lebih dari 100 dan berfluktuatif pada rentang

epoch 180 hingga 250. Dari hasil analisa *loss function* terhadap besar *epoch* dapat disimpulkan jika proses *training* untuk *epoch* lebih dari 100 tidak memberikan nilai yang signifikan pada hasil pendekatan fungsi. Maka penyelesaian model kandidat diputuskan menggunakan *epoch* kurang dari 200, yaitu pada rentang *epoch* di mana fluktuasi belum terjadi.

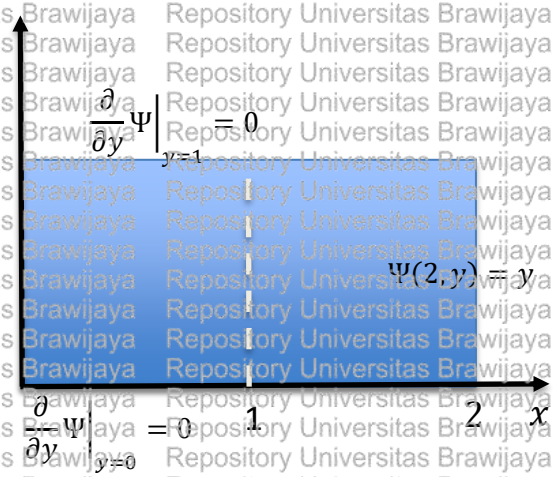
4.1.2 Persamaan *Laplace* (Syarat Batas *Neumann*)

Pada kasus kedua diterapkan suatu keadaan tunak difusi dalam *Laplacian* dengan syarat batas *Neumann* yang disketsakan pada Gambar 4.5. Persamaan *Laplace* tersebut memiliki rentang nilai x adalah [0,2] dan nilai y adalah [1,0] sebagai berikut:

$$\frac{\partial^2}{\partial x^2}\Psi(x,y) + \frac{\partial^2}{\partial y^2}\Psi(x,y) = 0 \qquad (4.16)$$

dengan syarat batas:

$$\begin{aligned} &\Psi(0,y) = 0 \\ &\Psi(2,y) = y \\ &\tfrac{\partial}{\partial y}\Psi = 0, pada\ y = 0\ dan\ y = 1 \end{aligned} \qquad (4.17)$$



Gambar 4.5. Sketsa kasus difusi dipersamaan (4.16) dengan keadaan syarat batas di persamaan (4.17).

Didapatkan solusi analitik (Barba & Forsyth, 2017) sebagai berikut:

$$\Psi(x,y) = \frac{x}{4} - 4\sum_{n=1,ganjil}^{\infty} \frac{1}{(n\pi)^2 \sinh(2n\pi)} \sinh(n\pi x)\cos(n\pi y) \quad (4.18)$$

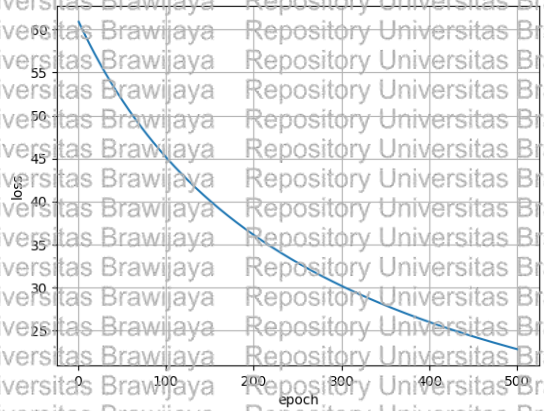
Maka solusi kandidat *neural network* digunakan syarat batas $B(x,y)$. Berikut solusi kandidat *neural network* yang digunakan:

$$\Psi_t(x,y) = B(x,y) + x(1-x)y\left[N(x,y,\vec{p}) - N(x,1,\vec{p}) - \frac{\partial}{\partial y}N(x,1,\vec{p})\right] \quad (4.19)$$

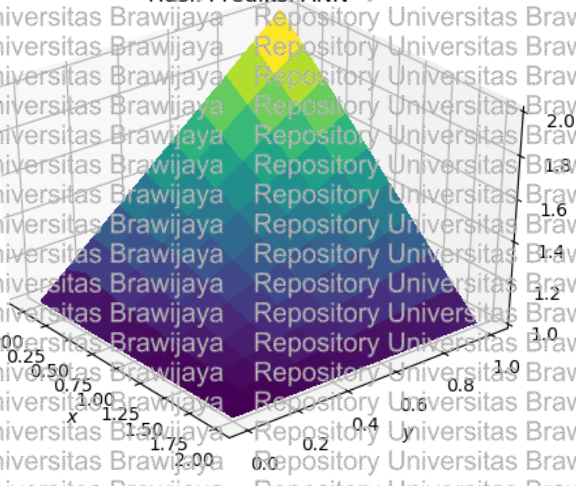
$$B(x,y) = x \cdot y$$

Hasil *training neural network* menunjukkan kurva interpolasi permukaan yang secara *visual* (Gambar 4.7) tidak mendekati hasil analitik (Gambar 4.8). Untuk mengetahui permasalahan tersebut, dilakukan peninjauan pada *starting point* parameter bobot dan peninjauan pada daerah konvergensi nilai *loss function*.

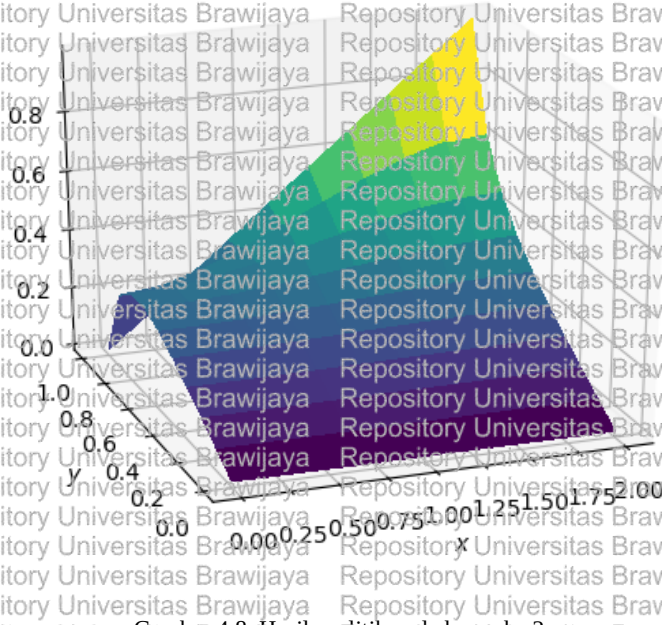
Peinjauan pertama dilakukan dari proses pra-pemrosesan pada parameter bobot. Penentuan bobot dengan mengatur random seed lebih besar dari 1 membuat nilai *loss* menjadi lebih besar dan bersifat *infinite*. Maka dari bobot awal ditentukan dengan random seed sebesar 1. Peninjauan berikutnya dilakukan dengan menganalisa nilai *loss function*. Proses percobaan training dilakukan sama dengan kasus pertama, yaitu dengan training 500 *epoch*. Pada *training* 500 *epoch*, nilai *loss* tidak menunjukkan konvergensi (gambar 4.4), maka untuk kasus kedua ini dilakukan *training* untuk *epoch* 1000 kali.



Gambar 4.6 Percobaan Training dengan epoch sebesar 500.



Gambar 4.7. Hasil prediksi persamaan *Laplace* dengan syarat batas *Neumann*



Gambar 4.8. Hasil analitik pada kasus ke 2

4.1.3 Persamaan Difusi 1 Dimensi (Syarat Batas Dirichlet)

Pada kasus berikut diselesaikan permasalahan kasus persamaan difusi 1 dimensi (Gambar 4.9) dengan kecepatan difusi yang dinyatakan pada persamaan berikut:

$$\frac{\partial U}{\partial t} = 0.003 \frac{\partial^2 U}{\partial x^2} \quad (4.20)$$

dengan syarat batas sebagi berikut:

$$U(0,t) = U(1,t) = 0$$
$$U(x,0) = 50x(1-x) \quad (4.21)$$

Diperoleh solusi analitik sebagai berikut (Lebl, 2019):

$$U(x,t) = \sum_{n=1,ganjil}^{\infty} \frac{400}{\pi^3 n^3} \sin(n\pi x) \exp -\pi^2 n^2 0.003t \quad (4.22)$$

Gambar 4.9. Sketsa keadaan koordinat kasus difusi 1 dimensi pada batang

Untuk solusi kandidat *neural network* digunakan solusi pada persamaan (4.4), dengan mengubah variabel untuk bagian y pada persamaan *Laplace* menjadi variabel t. Pemenuhan syarat batasnya diperoleh pada persamaan (4.3), dengan mensubtitusikan nilai syarat batas yang telah diketahui, sehingga diperoleh:

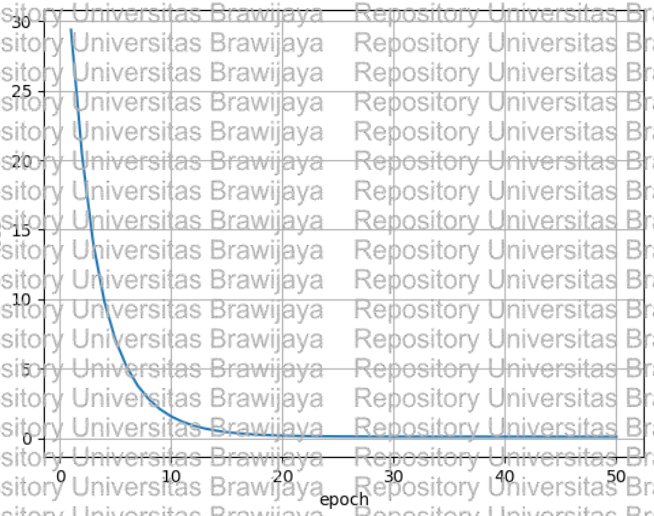
$$A(x,t) = (1-x) + (1-t)50x(1-x) + t(50x(1-x)) \quad (4.23)$$

Pada kasus ini, penghitungan *Loss function* dari persamaan umum *Laplace* dimodifikasi pada bagian diferensial terhadap

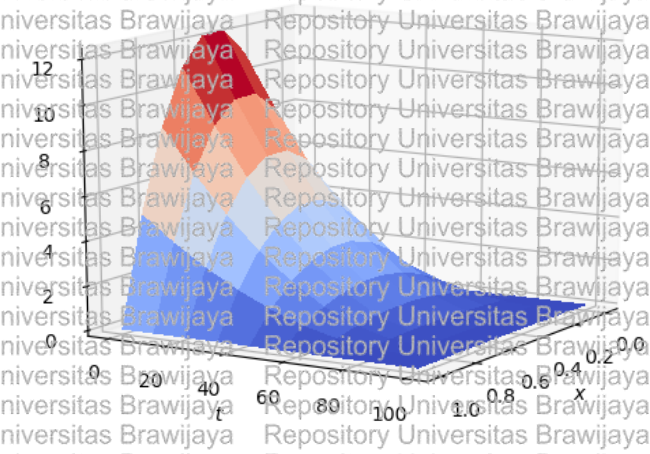
waktu. Mengacu pada persamaan (4.8), maka *Loss Function* dihitung dengan menerapkan persamaan berikut:

$$Loss[p] = \sum_i \left\{ \frac{\partial}{\partial t} \Psi(x_i, t_i) + \frac{\partial^2}{\partial^2 x} \Psi(x_i, t_i) - f_k(x_i, t_i) \right\}^2 \quad (4.24)$$

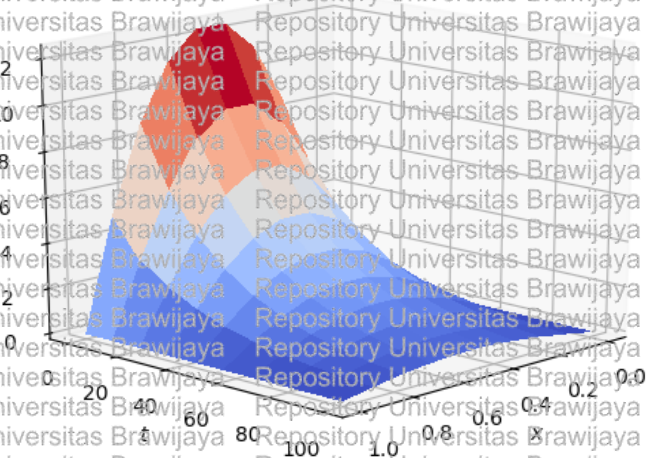
Untuk kasus difusi 1 dimensi ini, training dilakukan untuk menampilkan profil kejadian difusi dari $t = 0$ satuan hingga $t = 100$ posisi $x \in [0,1]$. Hasil tersebut menunjukkan proses *training* yang sangat baik karena pada *epoch* yang rendah dapat diperoleh profil prediksi difusi (Gambar 4.11) yang identik dengan profil penghitungan secara analitik (Gambar 4.12). Proses iterasi pada *epoch* dipilih hanya 10 kali karena pada analisis *epoch* sebanyak 50 kali menunjukkan saturasi di titik *epoch* ke 20 dan pengamatan pada interpolasi, profil dari prediksi difusi bernilai lebih tinggi dari hasil analitik untuk menghindari saturasi.



Gambar 4.10. Percobaan Training degan 50 epoch



Gambar 4.11. Hasil prediksi profil difusi 1 dimensi dengan konstanta difusi sebesar 0.003 pada t = 0 hingga t=100 sepanjang $x \in [0,1]$.



Gambar 4.12. Profil difusi 1 dimensi dengan konstanta difusi sebesar 0.003 pada t = 0 hingga t=100 sepanjang $x \in [0,1]$ yang dihitung secara analitik.

4.1.4 Persamaan Difusi 2 Dimensi

Diketahui kasus difusi dua dimensi pada suatu plat persegi $[0,1] \times [0,1]$ yang dinyatakan pada persamaan berikut:

$$c\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) = \frac{\partial T}{\partial t} \quad (4.25)$$

Dengan keadaan awal sebagai berikut:

$$T(x,y,0) = f(x,y), (x,y) \in R$$
$$R[0,1] \times [0,1] \quad (4.26)$$

Dan syarat batas:

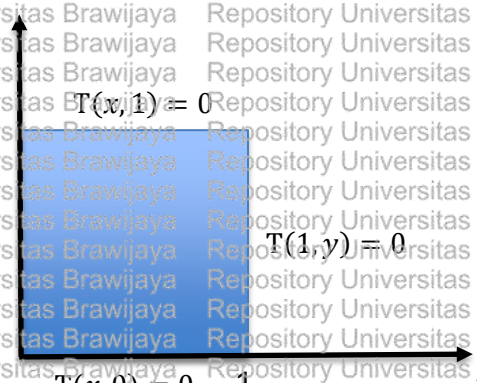
$$T(0,y,t) = T(a,y,t) = 0$$
$$T(x,0.t) = T(x,1,t) = 0 \quad (4.27)$$

Memiliki nilai konstanta difusi $c = \frac{1}{3}$, serta nilai $t = 3$.

$$f(x,y) = \begin{cases} 50 & jika\ y \leq 1 \\ 0 & jika\ y > 1 \end{cases}$$

Untuk kasus tersebut diperoleh solusi analitik sebagai berikut:

$$T(x,y,t) = \sum_{m=1}^{\infty} \sum_{n=1}^{\infty} \frac{800}{\pi^2}\left(-\cos\frac{m\pi}{2} + 1\right)\left(-\cos\frac{n\pi}{2} + 1\right)(\sin m\pi x)(\sin n\pi y)\left(e^{-\frac{\pi}{3}\sqrt{m^2+n^2}\cdot t}\right) \quad (4.28)$$



Gambar 4.13.Visualisasi keadaan syarat batas kasus 4.14 dari sisi permukaan atas

Cara penentuan kandidat model untuk menyelesaikan persamaan (4.25) yaitu dengan memperhatikan keadaan bagian koordinat ruang. Pada bagian ini, dilakukan modifikasi bentuk syarat batas dari penelitian Lagaris, dkk (1998) dengan menginterpolasi domain yang telah dibatasi oleh syarat batas yang telah ditentukan. Sehingga untuk mendapatkan data pada koordinat ruang dan isolasi keadaan batas, maka akan dicacah pada 4 bagian di koordinat rentang [0,1] yaitu pada bagian saat koordinat di posisi $x, y, x - 1, dan\ y - 1$. Maka syarat batasnya dimodelkan dengan:

$$A(x, y) = x \cdot (x - 1) \cdot y \cdot (y - 1) \tag{4.29}$$

Sedangkan untuk mengatasi perubahan T terhadap t waktu, faktor t akan dilibatkan pada model kandidat sebagai $t(t - 1)$ yang dinyatakan sebagai berikut:

$$T(x, y) = A(x, y) + x(1 - x)y(1 - y)t(t - 1)N(x, y, \vec{p}) \tag{4.28}$$

Pada kasus ke-empat dilakukan modifikasi dibagian penghitungan fungsi *loss* yaitu dengan melibatkan persamaan diferensial parsial yang diturunkan terhadap variabel t. Dari persamaan (4.25) bagian suku kiri persamaan yaitu $\frac{\partial T}{\partial t}$ dipindahkan ke ruas kanan, sehingga diperoleh nilai fungsi konstan di ruas kanan sebagai variabel pengontrol yaitu $f(x, y, t) = 0$, sehingga *loss function*-nya dinyatakan sebagai berikut:

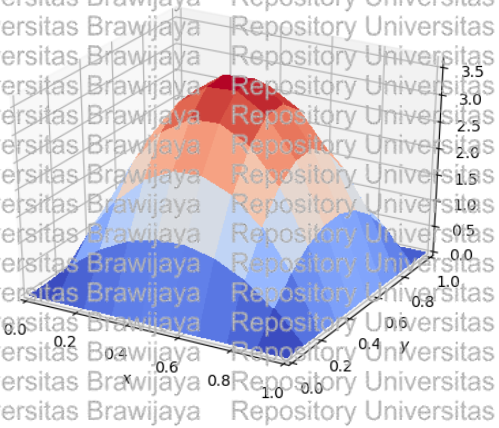
$$Loss[p] = \sum_i \left\{ \left[\left(\frac{\partial^2}{\partial x^2} T(x_i, y_i, t_i) + \frac{\partial^2}{\partial y^2} T(x_i, y_i, t_i) \right) c - \frac{\partial}{\partial t} T(x_i, y_i, t_i) \right] - f(x, y) \right\}^2 \tag{4.29}$$

Penghitungan nilai $\frac{\partial T}{\partial t}$ akan masuk ke dalam bagian dari *neural network*. Hal ini dimaksudkan untuk dapat mengontrol besar difusi pada syarat batas yang bergantung pada variabel waktu atau t.

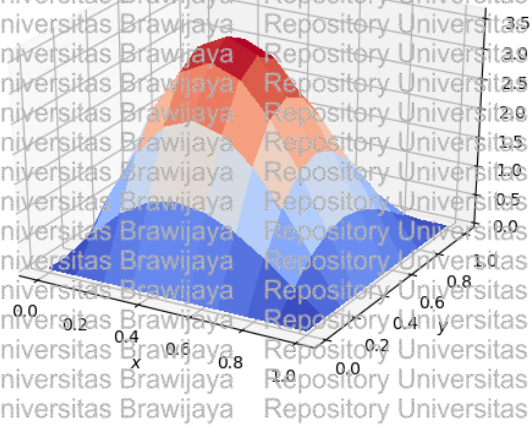
Hasil prediksi (Gambar 4.14) yang diperoleh secara grafik visual menunjukkan bahwa trend kurva interpolasi menyerupai *trend* kurva pada penghitungan eksak (Gambar 4.15) akan tetapi kurva

penghitungan eksak lebih ramping dibandingkan dengan kurva hasil prediksi.

Tinjauan berikutnya,pada proses *training* menunjukkan nilai saturasi pada *epoch* ke 20 dan menunjukkan nilai *loss/error* sebesar 20 yang tidak berubah (Gambar 4.16). Maka dalam hal ini dapat diduga jika perlu adanya modifikasi pada keadaan sayarat batas yang melibatkan domain *t*, agar diperoleh kandidat model yang tepat.



Gambar 4.14. Hasil visualisasi prediksi difusi pada t=3 dengan ANN, memperlihatkan trend yang sama dengan solusi numerik pada Gambar 4.15

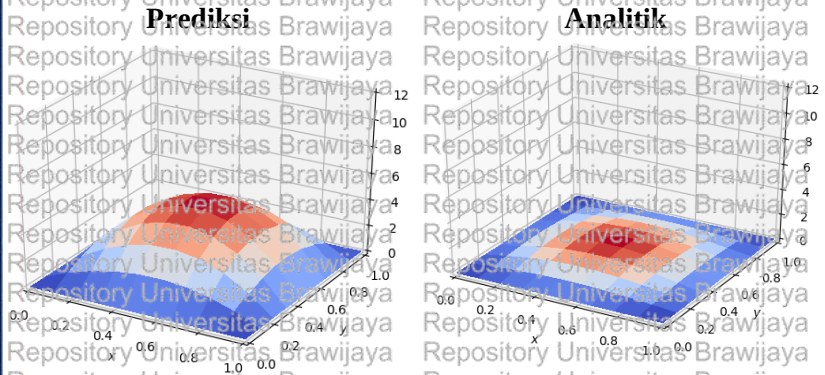


Gambar 4.15. Hasil visualisasi dari solusi analitik kasus ke-4

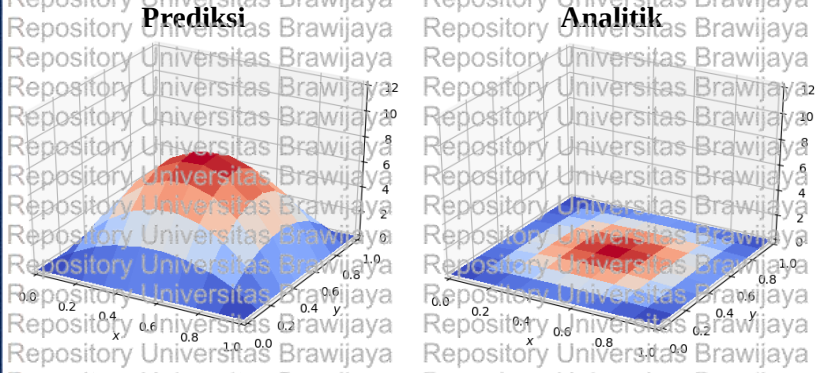
Gambar 4.16. Grafik perubahan nilai loss terhadap banyaknya epoch training untuk kasus ke-4

Gambar 4.17. Grafik perubahan nilai loss terhadap banyaknya *epoch training* untuk kasus ke-4, jika diamati maka keadaan saturasi mulai terjadi pada *epoch* ke-4

ari model kandidat final yang diperoleh dilakukan visualisasi difusi untuk nilai t yang berbeda yaitu pada $t = 4$ dan $t = 5$.



Gambar 4.18 Visualisasi prediksi difusi pada t=4, bagian kiri untuk hasil ANN dan kanan merupakan hasil numerik



Gambar 4.19. Visualisasi prediksi difusi untuk t=5, bagian kiri untuk hasil ANN dan kanan merupakan hasil numerik

4.2 Perbandingan Perhitungan Persamaan Diferensial Parsial Dengan Neural Network Terhadap Hasil Analitik

4.2.1 Kasus 1 (Persamaan Laplace dengan Syarat Batas Dirichlet)

Pada kasus pertama di persamaan (4.12), diperoleh hasil yang signifikan dalam perbandingan nilai antara kenaikan epoch dengan nilai RMSE (*Root Mean Squared Error*) prediksi terhadap hasil analitik. Hasil training menunjukkan jika semakin besar nilai *epoch*, maka nilai akurasi terhadap hasil eksak akan meningkat, hal ini ditunjukkan dari perbedaan nilai RMSE yang semakin kecil pada *epoch* besar. Akan tetapi pada *epoch* 500 diketahui nilai RMSE menjadi lebih besar dari nilai akurasi hasil numerik terhadap analitik. Maka dengan hasil perbandingan di Tabel 4.1, untuk endapatkan nilai pendekatan yang optimum proses training dilakukan pada batas *epoch* ke 100 dimana solusi kandidat dapat diterima.

Tabel 4.1. Perbandingan nilai RMSE hasil prediksi dan RMSE Numerik yang dibandingkan dengan solusi eksak

Epoch ke-	Loss	RMSE Prediksi ANN	RMSE Numerik
5	1759,0729	0,1597	0,1482
10	1661,5173	0,1539	0,1482
50	1539,2874	0,1459	0,1482
100	343,8970	0,0317	0,1482
500	229,8065	0,0716	0,1482

4.2.2 Kasus 2 (Persamaan Laplace dengan Syarat Batas Neumann)

Pada kasus kedua, hasil RMSE pada pendekatan neural network, memiliki selisih yang besar dibandingkan dengan RMSE pendekatan numerik. RMSE Prediksi ANN secara rata-rata berada pada nilai 1,04 dan tidak berubah secara signifikan ternadapa besarnya nilai epoch. Pada hasil pendekatan numerik, besar nilai RMSE numerik yaitu 1/10 dari RMSE Prediksi. Dari

perbandingan tersebut, pada kasus syarat batas *Neumann*, *neural network* tidak optimal dalam proses *training*-nya. Maka, perlu ada peninjauan pada cara penentuan syarat batas *Neumann* pada solusi kandidat.

Tabel 4.2. Perbandingan nilai RMSE hasil prediksi dan RMSE Numerik yang dibandingkan dengan solusi eksak pada kasus kedua.

Epoch ke-	Nilai Loss	RMSE Prediksi ANN	RMSE Numerik
5	206,6101	1,0426	0,1469
10	167,2293	1,0423	0,1469
50	99,7299	1,0418	0,1469
100	61,1156	1,0416	0,1469
500	22,8488	1,0413	0,1469

4.2.3 Kasus 3 (Persamaan Difusi 1 Dimensi dengan Syarat Batas Dirichlet)

Dalam kasus difusi satu dimensi, pada persamaan (4.12). perbandingan hasil antara pendekatan *neural network* dan persamaan eksak memberikan nilai RMSE yang lebih tinggi yaitu dengan selisih nilai setengah dari RMSE numerik. Nilai RMSE dari hasil prediksi terkecil yaitu pada proses training di *epoch* ke 5 yaitu sebesar 7,6228 dengan perbedaan sebesar 2,8769 dari RMSE numerik. Sehingga dapat ditentukan jika nilai yang baik digunakan untuk melakukan *training* pada model ini yaitu *epoch* lebih kecil, misalnya kurang dari 10.

Tabel 4.3. Perbandingan nilai RMSE hasil prediksi dan RMSE Numerik yang dibandingkan dengan solusi eksak pada kasus ketiga.

Epoch ke-	Loss	RMSE Prediksi ANN	RMSE Numerik
5	7,0439	7,6228	4,7459
10	0,4273	8,7292	4,7459
50	0,0959	12,3591	4,7459
100	0,0805	12,2263	4,7459
500	0,0418	11,7406	4,7459

4.2.4 Kasus 4 (Persamaan Difusi 2 Dimensi dengan syarat Batas Dirichlet)

Perbandingan hasil prediksi dengan *neural network* terhadap analitik untuk kasus difusi 2 dimensi menunjuk selisih nilai *error* terkecil pada *epoch* ke 100 yaitu sebesar 0,2807. Jika diamati dari *epoch* terkecil ke terbesar, terlihat bahwa besar *epoch* berbanding terbalik terhadap nilai *loss* dan nilai RMSE antara hasil prediksi dan solusi analitik. Maka pada kandidat model ini dapat terima untuk memecahkan kasus persamaan difusi dua dimensi. Akan tetapi perlu peninjauan ulang pada selisih perbedaan kurva dengan *epoch* yang lebih besar dan mengganti beberapa parameter untuk mencegah saturasi sebelum diperoleh hasil prediksi yang mendekati analitik.

Tabel 4.4. Perbandingan nilai error hasil prediksi terhadap solusi eksak dan error numerik terhadap solusi eksak pada kasus empat

epoch	Loss	Error ANN	Error Numerik
1	879,3709	0,9705	2,2534
10	122,3903	0,6067	2,2534
50	122,0878	0,5307	2,2534
100	119,8942	0,2435	2,2534

(Halaman ini sengaja dikosongkan)

BAB V
PENUTUP

5.1 Kesimpulan

Dari hasil penelitian ini diperoleh model *Artificial Neural Network* (ANN) yang dapat digunakan sebagai alternatif pencarian solusi persamaan diferensial parsial dengan arsitektur single hidden layer yang berisi 10 neuron dioptimasi pada parameter bobot. Aproksimsi terhadap solusi persamaan diferensial dengan ANN juga bergantung pada penentuan keadaan syarat batas, maka model solusi kandidat dibedakan menjadi dua yaitu solusi kandidat dengan syarat batas *Dirichlet* dan solusi trial dengan syarat batas Neumann. Pada syarat batas *Dirichlet*, akurasi model mendapatkan nilai yang lebih tinggi sebanding dengan metode numerik biasa, yaitu hanya berselisih ± 0.1 pada RMSE. Pada kasus persamaan difusi 3 dimensi yang melibatkan variabel x, y, dan t model kandidat berhasil mendekati nilai numerik dengan selisih rata-rata 0.5721. Akan tetapi, pada solusi trial dengan keadaan *Neumann*, nilai akurasi RMSE berselisih 10 kali lebih besar dari hasil RMSE numerik.

5.2 Saran

Pada penelitian berikutnya diharapkan adanya solusi kandidat untuk kasus lebih dari dua dimensi dan varisi terhadap bentuk keadaan syarat batas selain dari dua keadaan syarat batas pada penelitian ini.

(Halaman ini sengaja dikosongkan)

Repository Universitas Brawijaya
UNIVERSITAS BRAWIJAYA
REPOSITORY.UB.AC.ID

DAFTAR PUSTAKA

Al-Aradi, A. of T., Correia, A. (Instituto de M. P. e A., Naiff, D. (Universidade F. do R. de J., & Jardim, G. (Fundacao G. V. (2018). *Solving Nonlinear and High-Dimensional Partial Differential Equations via Deep Learning*.

Arfken, G. B., Weber, H. J., & Harris, F. E. (2013). *Mathematical Methods for Physicists* (Seventh). Oxford: Elsevier Inc.

Barba, L. A., & Forsyth, G. F. (2017). 12 Steps to Navier-Stokes. Retrieved February 2, 1BC, from https://nbviewer.jupyter.org/github/barbagroup/CFDPython/blob/master/lessons/12_Step_9.ipynb

Boyce, W. E., & DiPrima, R. C. (2001). *Elementary Differential Equations and Boundary Value Problems* (7th Editio). New York: John Wiley & Sons, Inc.

Chiaramonte, M. M., & Kiener, M. (2013). Solving differential equations using neural networks, *1*(4), 1–5.

Dougal Maclaurin, David Duvenaud, Matt Johnson, J. T. (2018). Autograd. Retrieved February 2, 2019, from https://github.com/HIPS/autograd

Giordano, N. J., & Nakanishi, H. (2006). *Computational Physics* (2nd ed.). Canada: Pearson Education, Inc.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Hancock, M. J. (2006). *The 1-D Heat Equation The 1-D Heat Equation*.

Hayati, M., & Karami, B. (2007). Feedforward Neural Network for Solving Partial Differential Equations. *Journal of Applied Sciences*, *7*(19), 2812–2817.

Honchar, A. (2017). Neural Networks for Solving Differential Equations. Retrieved February 28, 2019, from https://becominghuman.ai/neural-networks-for-solving-differential-equations-fa230ac5e04c

agaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, *9*(5), 987–1000. https://doi.org/10.1109/72.712178

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

aissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations, (Part I), 1–22. Retrieved from https://arxiv.org/abs/1711.10561

ena, S. (2017). Pengenalan Deep Learning Part 3 : BackPropagation Algorithm. Retrieved May 3, 2019, from https://medium.com/@samuelsena/pengenalan-deep-learning-part-3-backpropagation-algorithm-720be9a5fbb8

irignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, *375*, 1339–1364. https://doi.org/10.1016/J.JCP.2018.08.029

eo, K., & Melnyk, I. (2019). Deep learning algorithm for data-driven simulation of noisy dynamical system. *Journal of Computational Physics*, *376*, 1212–1231. https://doi.org/10.1016/j.jcp.2018.10.024

accone, G. (2016). *Getting Started with TensorFlow*. Birmingham: Packt Publishing Ltd.

Lampiran 1. Grafik Visual Hasil Training

Interpolasi hasil training untuk kasus ke 1

Gambar 1. Epoch ke-5

Gambar4. Epoch ke 100

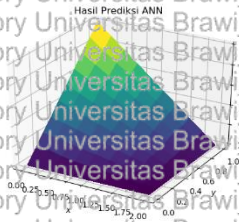
Gambar 2. Epoch ke-10

Gambar5. Epoch ke-500

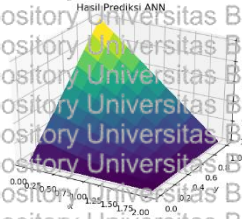
Gambar3. Epoch ke-50

nterpolasi hasil training untuk kasus ke 2

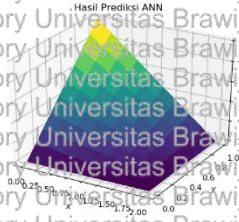
Gambar 1. Epoch ke-10



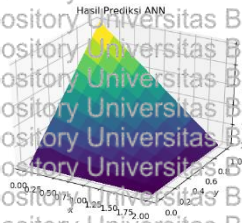
Gambar 4. Epoch ke-50



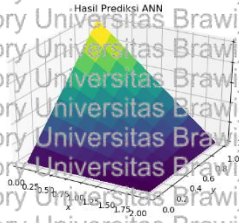
Gambar 2. Epoch ke-5



Gambar 5. Epoch ke 100

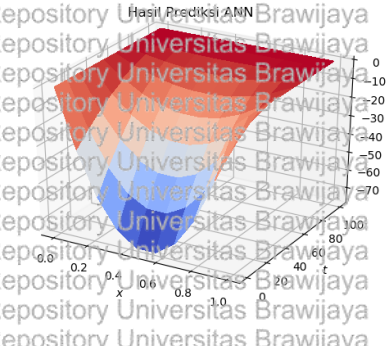


Gambar 3. Epoch ke-10

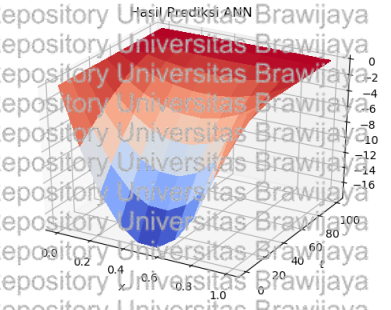


Gambar 6. Epoch ke-500

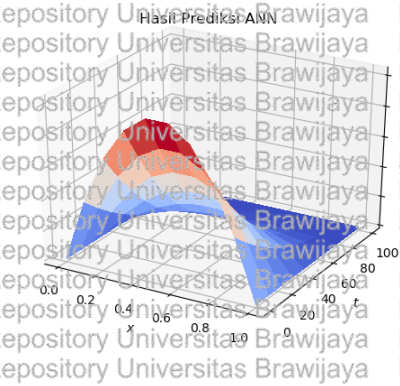
Interpolasi hasil training untuk kasus ke 3



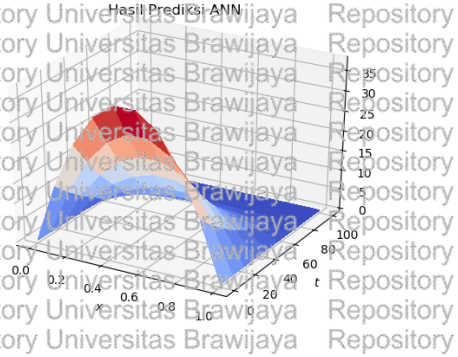
Gambar 7. Epoch ke-1



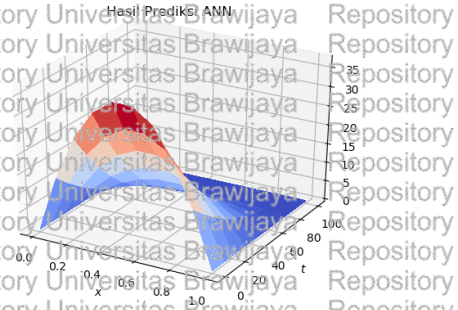
Gambar 8. Epoch ke-5



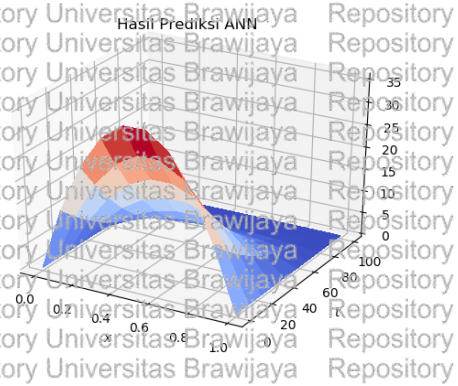
Gambar 9. Epoch ke-10



Gambar 10. Epoch ke-50



Gambar 11. Epoch ke 100



Gambar 12. Epoch ke-500

(Halaman ini sengaja dikosongkan)

Repository Universitas Brawijaya

Lampiran 2. Kode Program

Kode program Kasus 1

```
import matplotlib.pyplot as plt
import autograd.numpy as np
from autograd import grad, jacobian
import autograd.numpy.random as npr
from matplotlib import pyplot, cm
import csv
from sklearn.metrics import mean_squared_error
np.random.seed(5)
from mpl_toolkits.mplot3d import Axes3D

### Neural Network ###
def f(x):
    out = 0.
    return out

def relu(x):
    out = np.maximum(0, x)
    return out

def sigmoid(x):
    out = 1. / (1. + np.exp(-x))
    return out

def neural_network(W, x, b):
    a1 = sigmoid(np.dot(x, W[0]) + b[0])
    out = np.dot(a1, W[1]) + b[1]
    return out

def neural_network_x(W, x, b):
    a1 = sigmoid(np.dot(x, W[0]) + b[0])
    out = np.dot(a1, W[1]) + b[1]
    return out

def A(x):
    #out = x[1] * np.sin(np.pi * x[0])
    #out = x[0] * 2 * x[1]
    #out = x[0] * x[1]
    out = x[1] * np.sin(np.pi * x[0])
    return out

def psy_trial(x, net_out):
```

```
    out = A(x) + x[0] * (1 - x[0]) * x[1] * (1 - x[1]) *
et_out
    return out

ef loss_function(W, x, y, b):
    loss_sum = 0.
    for xi in x:
        for yi in y:
            input_point = np.array([xi, yi])
            net_out = neural_network(W, input_point,
)[0]

            psy_t_hessian =
acobian(jacobian(psy_trial))(input_point, net_out)

            gradient_of_trial_d2x = psy_t_hessian[0][0]
            gradient_of_trial_d2y = psy_t_hessian[1][1]

            func = f(input_point)

            err_sqr = ((gradient_of_trial_d2x +
radient_of_trial_d2y) - func)**2
            loss_sum = loss_sum + err_sqr
    return loss_sum

ef analytic_solution(x):
    return (1 / (np.exp(np.pi) - np.exp(-np.pi))) * \
           np.sin(np.pi * x[0]) * (np.exp(np.pi * x[1]) -
np.exp(-np.pi * x[1]))

ef plot2D(x, y, p):
    fig = pyplot.figure()
    ax = fig.gca(projection='3d')
    X, Y = np.meshgrid(x, y)
    surf = ax.plot_surface(X, Y, p[:], rstride=1,
stride=1, cmap=cm.viridis,
                           linewidth=0, antialiased=False)
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.set_zlim(0, 1)
    ax.set_xlabel('$x$')
    ax.set_ylabel('$y$')
    plt.savefig('numerik-case-1.png')
    plt.close()

ef lapiace2d(p, y, dx, dy, l1norm_target):
8
```

Repository Universitas Brawijaya
UNIVERSITAS BRAWIJAYA
REPOSITORY.UB.AC.ID

```
    l1norm = 1
    pn = np.empty_like(p)

    while l1norm > l1norm_target:
        pn = p.copy()
        p[1:-1, 1:-1] = ((dy ** 2 * (pn[1:-1, 2:] +
pn[1:-1, 0:-2]) + dx ** 2 * (pn[2:, 1:-1] + pn[0:-2, 1:-
1])) / (2 * (dx ** 2 + dy ** 2)))
        p[:, 0] = 0  # p = 0 @ x = 0
        p[:, -1] = np.sin(np.pi * y) * 0.5    # p = y @ x
= 1
        p[0, :] = 0 # p = 0 @ y = 0
        p[-1, :] = 0    # p = sin pi*x @ y=1
        l1norm = (np.sum(np.abs(p[:]) - np.abs(pn[:])) /
                  np.sum(np.abs(pn[:])))
    return p

def main(a):
    #plate size, mm
    w = h = 1
    #intervals in x-, y-, directions, mm
    dx = dy = 0.1
    #thermal diffusivity of steel, mm2.s-1
    p = 4

    nx, ny = 10, 10
    x_space = np.linspace(0, w, nx)
    y_space = np.linspace(0, h, ny)
    # variabel declaration

    nx, ny = 10, 10
    c = 0
    dx = 2. / (nx - 1)
    dy = 2. / (ny - 1)

    #Neural Network adjustable parameter
    W = [npr.rand(2, 10), npr.rand(10, 1)]
    #b = [npr.rand(np.size(W[0][0])),
npr.rand(np.size(W[1]))]
    b = [np.zeros(np.size(W[0][0])),
np.zeros(np.size(W[1]))]

    learning_rate = 0.01

    #print neural_network(W, np.array([1, 1]), b[[1, 1]])
```

```
    print("init weight...")
    loss = []
    epoch = []
    iter = 0
    for i in range(a):
        print('%d' % i)
        L = loss_function(W, x_space, y_space, b)
        loss.append(L)

        iter = iter + 1
        epoch.append(iter)

        loss_grad = grad(loss_function)(W, x_space,
space, b)
        W[0] = W[0] - learning_rate * loss_grad[0]
        W[1] = W[1] - learning_rate * loss_grad[1]
        #b[0] = b[0] - learning_rate * loss_grad[0]
        #b[1] = b[1] - learning_rate * loss_grad[0]

        print loss[i]

    surface2 = np.zeros((ny, nx))

    print("neural net solution...")
    for i, x in enumerate(x_space):
        for j, y in enumerate(y_space):
            net_outt = neural_network(W, [x, y], b)[0]
            surface2[i][j] = psy_trial([x, y], net_outt)

    '''Neural Network Solution'''
    fig1 = plt.figure()
    ax = fig1.gca(projection='3d')
    X, Y = np.meshgrid(x_space, y_space)
    surf = ax.plot_surface(X, Y, surface2, rstride=1,
stride=1, cmap=cm.viridis,
                           linewidth=0,
ntialiased=False)

    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.set_zlim(0, 2)

    ax.set_title('Prediksi ANN')
    ax.set_xlabel('$x$')
    ax.set_ylabel('$y$')
    plt.savefig('gambar-dengan-epoch-{}.png'.format(a))
    plt.close()
```

Repository Universitas Brawijaya
UNIVERSITAS BRAWIJAYA
REPOSITORY.UB.AC.ID

```
    surface_analytic = np.zeros((ny, nx))
    '''Solusi Analitik'''
    for i, x in enumerate(x_space):
        for j, y in enumerate(y_space):
            surface_analytic[i][j] = analytic_solution([x, y])
    fig2 = plt.figure()
    ax = fig2.gca(projection='3d')
    X, Y = np.meshgrid(x_space, y_space)
    surf = ax.plot_surface(X, Y, surface_analytic, rstride=1, cstride=1, cmap=cm.viridis,
                           linewidth=0, antialiased=False)
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.set_zlim(0, 2)
    ax.set_title('Solusi Analitik')
    ax.set_xlabel('$x$')
    ax.set_ylabel('$y$')
    plt.savefig('solusi-analitik-{}.png'.format(a))
    plt.close()

    fig3 = plt.figure()
    plt.title('Loss Function Kasus1')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.plot(epoch, loss)
    plt.grid()
    plt.savefig('loss-epoch-{}.png'.format(a))
    plt.close()

    '''Solusi Numerik'''
    ##initial conditions
    p = np.zeros((ny, nx))  # create a XxY vector of 0's
    p = laplace2d(p, x_space, dx, dy, 1e-4)
    plot2D(x_space, y_space, p)

    with open('RMSEcase1-{}-epoch.csv'.format(a), 'w',) as csvFile:
        writer = csv.writer(csvFile, delimiter=',',
                            lineterminator='\n',)
```

```
            delta_ANN =
p.sqrt(mean_squared_error(surface_analytic, surface2))
            delta_Numerik =
p.sqrt(mean_squared_error(surface_analytic, p))
            writer.writerow([ epoch , 'RMSE_ann_Analitik',
'RMSE_Numerik_Analitik', loss ])
            writer.writerow(['{}'.format(a), delta_ANN,
delta_Numerik, loss[a-1]])
            print 'nilai RMSE', delta_ANN

= [5, 10, 50, 100, 500]
or a in x:
    main(a)
```

Kode program Kasus 2

```
'''Solving Laplace Equation with PDE'''
import matplotlib.pyplot as plt
import autograd.numpy as np
from autograd import grad, jacobian
import autograd.numpy.random as npr
from matplotlib import pyplot, cm
np.random.seed(5)
from mpl_toolkits.mplot3d import Axes3D

def plot2D(x, y, p):
    fig = pyplot.figure(figsize=(11, 7), dpi=100)
    ax = fig.gca(projection='3d')
    X, Y = np.meshgrid(x, y)
    surf = ax.plot_surface(X, Y, p[:], rstride=1,
cstride=1, cmap=cm.viridis,
                           linewidth=0, antialiased=False)
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.view_init(30, 225)
    ax.set_xlabel('$x$')
    ax.set_ylabel('$y$')
    #plt.show()

def laplace2d(p, y, dx, dy, l1norm_target):
    l1norm = 1
    pn = np.empty_like(p)

    while l1norm > l1norm_target:
        pn = p.copy()
        p[1:-1, 1:-1] = ((dy ** 2 * (pn[1:-1, 2:] +
pn[1:-1, 0:-2]) + dx ** 2 * (pn[2:, 1:-1] + pn[0:-2, 1:-
1])) / (2. * (dx ** 2 + dy ** 2)))
        p[:, 0] = 0 # p = 0 @ x = 0
        p[:, -1] = y # p = y @ x = 2
        p[0, :] = p[1, :]  # dp/dy = 0 @ y = 0
        p[-1, :] = p[-2, :]  # dp/dy = 0 @ y = 1
        l1norm = (np.sum(np.abs(p[:]) - np.abs(pn[:])) /
                  np.sum(np.abs(pn[:])))
    return p

#variabel declaration
x = 1
y = 1
```

```
x, ny = 10, 10
 = 1
x = 2. / (nx - 1)
y = 2. / (ny - 1)

_space = np.linspace(0, x, nx)
_space = np.linspace(0, y, ny)

#initial conditions
 = np.zeros((ny, nx))  # create a XxY vector of 0's

#boundary conditions
[:, 0] = 0  # p = 0 @ x = 0
[:, -1] = y_space  # p = y @ x = 2
[0, :] = p[1, :]  # dp/dy = 0 @ y = 0
[-1, :] = p[-2, :]  # dp/dy = 0 @ y = 1

 = laplace2d(p, y_space, dx, dy, 1e-4)
lot2D(x_space, y_space, p)

## Neural Network ###

ef f(x):
    out = 0
    return out

ef sigmoid(x):
    out = 1. / (1. + np.exp(-x))
    return out

ef neural_network(W, x):
    a1 = sigmoid(np.dot(x, W[0]))
    out = np.dot(a1, W[1])
    return out

ef neural_network_x(x):
    a1 = sigmoid(np.dot(x, W[0]))
    out = np.dot(a1, W[1])
    return out

ef B(x):
    out = x[0] * x[1] + x[1] * np.sin(np.pi * x[0])
    return out

ef psy_trial(x, net_out):
    out = B(x) + x[0] * (1 - x[0]) * x[1] * net_out
    return out

ef loss_function(W, x, y):
    
4
```

Repository Universitas Brawijaya
UNIVERSITAS BRAWIJAYA
REPOSITORY.UB.AC.ID

```
    loss_sum = 0
    for xi in x:
        for yi in y:
            input_point = np.array([xi, yi])
            input_point_x1 = np.array([xi, 1])
            net_out_xy = neural_network(W,
input_point)[0]
            net_out_x1 = neural_network(W,
input_point_x1)[0]
            dnet_out_x1 = grad(neural_network)(W,
input_point)[1][1]
            net_out = net_out_xy + net_out_x1 *
dnet_out_x1
            psy_t_hessian =
jacobian(jacobian(psy_trial))(input_point, net_out)[0]
            gradient_of_trial_d2x = psy_t_hessian[0][0]
            gradient_of_trial_d2y = psy_t_hessian[1][1]
            func = f(input_point)
            err_sqr = ((gradient_of_trial_d2x +
gradient_of_trial_d2y) - func) ** 2
            loss_sum = loss_sum + err_sqr
    return loss_sum

W = [npr.rand(2, 10), npr.randn(10, 1)]
learning_rate = 0.01
print neural_network(W, np.array([1, 1]))
loss = []
epoch = []
iter = 0
print("init weight...")
for i in range(100):
    print("%d : " % i)
    lossval = loss_function(W, x_space, y_space)
    loss.append(lossval)
    iter = iter + 1
    epoch.append(iter)
    loss_grad = grad(loss_function)(W, x_space, y_space)
    W[0] = W[0] - learning_rate * loss_grad[0]
    W[1] = W[1] - learning_rate * loss_grad[1]
    print loss[i]

print loss_function(W, x_space, y_space)
```

```
urface2 = np.zeros((ny, nx))

rint("neural net solution...")
or i, x in enumerate(x_space):
    for j, y in enumerate(y_space):
        dif_net_out_x1 = grad(neural_network)(W, [x,
])[1][1]
        net_outt = (neural_network(W, [x, y]) +
eural_network(W, [x, 1]) - dif_net_out_x1)[0]
        surface2[i][j] = psy_trial([x, y], net_outt)

lt.plot(epoch[10:], loss[10:])
lt.xlabel('epoch')
lt.ylabel('loss')
lt.grid(b=None, which='both', axis='both')
lt.show()

ig = plt.figure()
x = fig.gca(projection='3d')
, Y = np.meshgrid(x_space, y_space)
urf = ax.plot_surface(X, Y, surface2, rstride=1,
stride=1, cmap=cm.viridis,
                      linewidth=0, antialiased=False)

x.set_xlabel('$x$')
x.set_ylabel('$y$')
lt.show()
```

Kode Program Kasus 3

```
import matplotlib.pyplot as plt
import autograd.numpy as np
from autograd import grad, jacobian
import autograd.numpy.random as npr
from matplotlib import pyplot, cm
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
import csv
np.random.seed(50)
from mpl_toolkits.mplot3d import Axes3D

### Neural Network ###

def f(x):
    out = 0.
    return out

def sigmoid(x):
    out = 1. / (1. + np.exp(-x))
    return out

def neural_network(W, x, b):
    a1 = sigmoid(np.dot(x, W[0]) + b[0])
    out = np.dot(a1, W[1]) + b[1]
    return out

def neural_network_x(W, x, b):
    a1 = sigmoid(np.dot(x, W[0]) + b[0])
    out = np.dot(a1, W[1]) + b[1]
    return out

def A(x):
    out = (1 - x[0]) + (1 - x[1]) * 50 * x[0] * (1 -
x[0]) + x[1] * (50 * x[0] * (1 - x[0]))
    return out

def psy_trial(x, net_out):
    out = A(x) * 0.003 + x[0] * (1 - x[0]) * x[1] * (1 -
x[1]) * net_out
    return out

def loss_function(W, x, y, b):
    loss_sum = 0.
    D = 0.003
```

```
    for xi in x:
        for yi in y:
            input_point = np.array([xi, yi])
            net_out = neural_network(W, input_point,
)[0]
            psy_t_hessian =
acobian(jacobian(psy_trial))(input, net_out)
            gradient_of_trial_d2x = psy_t_hessian[0][0]
            gradient_of_trial_dt =
acobian(psy_trial)(input, net_out)[0]
            func = f(input_point)
            err_sqr = (gradient_of_trial_dt - D *
radient_of_trial_d2x - func)**2
            loss_sum = loss_sum + err_sqr
    return loss_sum

 = 0.003
 = 1000
ef analytic_solution(x, t):
    analitik = 0
    for i in range(1, n, 2):
        #sum = 400 / ((np.pi**3) * (2.0 * n - 1)) *
p.sin((2 * n - 1) * np.pi * x)
        #ganjil =(2 * n) - 1
        a = 400.0 / ((np.pi ** 3) * (i ** 3))
        b = np.sin((i * np.pi * x))
        c = np.exp(-D * (i ** 2) * (np.pi ** 2) * t)
        sum = a * b * c
        analitik += sum
    return analitik

x = 10
t = 10
 = 1

_space = np.linspace(0, w, nx)
_space = np.linspace(0, 10, nt)

 = [npr.rand(2, 10), npr.rand(10, 1)]
 = [np.zeros(np.size(W[0][0])), np.zeros(np.size(W[1]))]
nput = np.asarray(x_space)

ef main(a):
    t = 100
    tt_space = np.linspace(0, t, nt)
```

```
loss = []
epoch = []
iter = 0
learning_rate = 0.01
for i in range(a):
    print('i = ', i)
    L = loss_function(W, x_space, t_space, b)
    loss.append(L)
    iter = iter + 1
    epoch.append(iter)
    loss_grad = grad(loss_function)(W, x_space,
t_space, b)
    W[0] = W[0] - learning_rate * loss_grad[0]
    W[1] = W[1] - learning_rate * loss_grad[1]
    #b[0] = b[0] - learning_rate * loss_grad[0]
    #b[1] = b[1] - learning_rate * loss_grad[0]
    print loss[i]

'''Plot Analitik'''
Z = np.zeros((nx, nt))
for i, t in enumerate(tt_space):
    for j, x in enumerate(x_space):
        Z[i][j] = analytic_solution(x, t)

fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y = np.meshgrid(x_space, tt_space)
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
linewidth=0, antialiased=False)
ax.set_title('Solusi Analitik')
ax.set_xlabel('$x$')
ax.set_ylabel('$t$')
plt.savefig('analitik-case2-{}.png'.format(a))
plt.close()

''' Plot NN '''
surface = np.zeros((nx, nt))
for i, t in enumerate(t_space):
    for j, x in enumerate(x_space):
        net_out_result = neural_network(W, [x, t],
b)[0]
        surface[i][j] = psy_trial([x, t],
net_out_result)
```

```
    tt_space = np.linspace(0, 100, 10)
    X, Y = np.meshgrid(x_space, np.flip(tt_space))
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X, Y, surface,
map=cm.coolwarm, linewidth=0, antialiased=False)
    ax.set_title('Hasil Prediksi ANN')
    ax.set_xlabel('$x$')
    ax.set_ylabel('$t$')
    plt.savefig('ANN-case2-{}.png'.format(a))
    plt.close()

    plt.plot(epoch, loss)
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.grid()
    plt.savefig('loss-case2-epoch-{}.png'.format(a))
    plt.close()

    #Numerik Solution
    M = 10  # GRID POINTS on space interval
    N = 10  # GRID POINTS on time interval

    x0 = 0
    xL = 1

    # ----- Spatial discretization step -----
    dx = (xL - x0) / (M - 1.)

    t0 = 0.
    tF = 1

    # ----- Time step -----
    dt = (tF - t0) / (N - 1)

    D = 0.003  # Diffusion coefficient
    alpha = -3.  # Reaction rate

    r = dt * D / dx ** 2
    s = dt * alpha;

    print("r=",r)

    # ----- Creates grids -----
    xspan = np.linspace(x0, xL, M)
    tspan = np.linspace(t0, tF, N)

    # ----- Initializes matrix solution U -----
    U = np.zeros((M, N))
```

```
    # ----- Initial condition -----
    U[:, 0] = 50. * xspan * (1 - xspan)

    # ----- Dirichlet Boundary Conditions -----
    U[0, :] = 0.0
    U[-1, :] = 0.0

    # ----- Equation (15.8) in Lecture 15 -----
    for k in range(0, N - 1):
        for i in range(1, M - 1):
            U[i, k + 1] = r * U[i - 1, k] + (1 - 2 * r + s) * U[i, k] + r * U[i + 1, k]

    X, T = np.meshgrid(tspan, xspan)

    fig = plt.figure()
    ax = fig.gca(projection='3d')

    surf = ax.plot_surface(X, T, U, cmap=cm.coolwarm,
                           linewidth=0, antialiased=False)

    ax.set_xlabel('t')
    ax.set_ylabel('x')
    ax.set_zlabel('U')
    plt.savefig('Numerik-case2-{}.png'.format(a))
    plt.close()

    with open('RMSEcase2-{}epoch.csv'.format(a), 'w', as csvFile:
        writer = csv.writer(csvFile, delimiter=',',
lineterminator='\n',)
        delta_ANN = np.sqrt(mean_squared_error(Z, surface))
        delta_Numerik = np.sqrt(mean_squared_error(Z, U))
        writer.writerow(['epoch', 'RMSE Ann Analitik',
'RMSE Numerik Analitik', 'loss'])
        writer.writerow([{}.format(a), delta_ANN,
delta_Numerik, loss[a-1]])
        print('Nilai RMSE', delta_ANN

x = [10]
for a in x:
    main(a)
```

Code Program Kasus 4

```
import matplotlib.pyplot as plt
import autograd.numpy as np
from autograd import grad, jacobian
import autograd.numpy.random as npr
from matplotlib import pyplot, cm
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import csv
np.random.seed(5)
from mpl_toolkits.mplot3d import Axes3D

nx = 10
ny = 10
nt = 10
nu = .05
dx = 2. / (nx - 1.)
dy = 2. / (ny - 1.)
sigma = .25
dt = sigma * dx * dy / nu
t = 0.1

### Analitik ###
n = 3
m = 100
def part_a(m, n):
    out = (1 - np.cos((m * np.pi) / 2)) * (1 - np.cos((n
 np.pi) / 2))
    return out

def part_b(m, n, x, y):
    out = np.sin(m * np.pi * x) * np.sin(n * np.pi * y)
    return out

def part_exp(m, n, t):
    out = np.exp(-np.pi / 3 * np.sqrt(m**2 * n**2) * t)
    return out

def analytic(x, y):
    solusi = 0
    for i in range(n):
        for j in range(m):
```


Repository Universitas Brawijaya
REPOSITORY.UB.AC.ID
UNIVERSITAS BRAWIJAYA

```
                iter = (800. / np.pi**2) * part_a(i, j) *
part_b(i, j, x, y) * part_exp(i, j, t)
                solusi += iter
    return solusi

### Neural Network ###
def f(x):
    out = 0.
    return out

def sigmoid(x):
    out = 1. / (1. + np.exp(-x))
    return out

def neural_network(W, x, b):
    a1 = sigmoid(np.dot(x, W[0]) + b[0])
    out = np.dot(a1, W[1]) + b[1]
    return out

def neural_network_x(W, x, b):
    a1 = sigmoid(np.dot(x, W[0]) + b[0])
    out = np.dot(a1, W[1]) + b[1]
    return out

def A(x):
    out = x[1] * x[0] * (1-x[1]) * (1-x[0])
    return out

def psy_trial(x, net_out):
    out = A(x) + x[0] * (1 - x[0]) * x[1] * (1 - x[1]) *
x[2] * (1 - x[2]) * net_out
    return out

def loss_function(W, x, y, t, b):
    loss_sum = 0.
    for ti in t:
        for xi in x:
            for yi in y:
                input_point = np.array([xi, yi, ti])
                net_out = neural_network(W, input_point,
b)[0]
                psy_t_hessian =
jacobian(jacobian(psy_trial))(input_point, net_out)
                psy_t_jacobian
=jacobian(psy_trial)(input_point, net_out)
```

```
                gradient_of_trial_d2x = psy_t_hessian[0][0]
                gradient_of_trial_d2y = psy_t_hessian[1][1]
                gradient_of_trial_dt = psy_t_jacobian[2]
                func = f(input_point)
                err_sqr = ((((gradient_of_trial_d2x + gradient_of_trial_d2y) * 1/3 - gradient_of_trial_dt) - func)**2)
                loss_sum = loss_sum + err_sqr
    return loss_sum

def main(a):
    #plate size, mm
    w = h = 1
    #intervals in x-, y-, directions, mm
    dx = dy = 0.1
    #thermal diffusivity of stell, mm2.s-1
    D = 4

    nx, ny, nt = 10, 10, 10

    x_space = np.linspace(0, w, nx)
    y_space = np.linspace(0, h, ny)
    t_space = np.linspace(0, 3, nx)
    t_space = t_space.reshape(-1, 1)

    scaler = MinMaxScaler(feature_range=(0, 1))
    rescaledT = scaler.fit_transform(t_space)

    np.set_printoptions(precision=3)

    W = [npr.rand(3, 10), npr.rand(10, 1)]
    #b = [npr.rand(np.size(W[0][0])), npr.rand(np.size(W[1]))]
    b = [np.zeros(np.size(W[0][0])), np.zeros(np.size(W[1]))]

    learning_rate = 0.01
    #print neural_network(W, np.array([1, 1]), b([1, 1]))
```

```
    print("init weight...")
    loss = []
    epoch = []
    iter = 0
    for i in range(a):
        print('%d' % i)
        L = loss_function(W, x_space, y_space, rescaledT,
b)
        loss.append(L)
        iter = iter + 1
        epoch.append(iter)
        loss_grad = grad(loss_function)(W, x_space,
y_space, rescaledT, b)
        W[0] = W[0] - learning_rate * loss_grad[0]
        W[1] = W[1] - learning_rate * loss_grad[1]
        #b[0] = b[0] - learning_rate * loss_grad[0]
        #b[1] = b[1] - learning_rate * loss_grad[0]
        print loss[i]
    surface2 = np.zeros((ny, nx))
    print("neural net solution...")
    #t = 0
    inverseScaleT = scaler.inverse_transform(rescaledT)
    for ti, t in enumerate(inverseScaleT):
        for i, x in enumerate(x_space):
            for j, y in enumerate(y_space):
                net_outt = neural_network(W, [x, y, t],
b)[0]
                surface2[i][j] = psy_trial([x, y, t],
net_outt) * 2
    fig1 = plt.figure()
    ax = fig1.gca(projection='3d')
    X, Y = np.meshgrid(x_space, y_space)
    surf = ax.plot_surface(X, Y, surface2, rstride=1,
cstride=1, cmap=cm.coolwarm,
                           linewidth=0,
antialiased=False)
    ax.set_xlim(0, 1)
```

```
    ax.set_ylim(0, 1)
    #ax.set_zlim(0, 2)

    ax.set_xlabel('$x$')
    ax.set_ylabel('$y$')
    plt.savefig('gambar-case4-mse-dengan-iterasi-{}.png'.format(a))
    plt.close()
    plt.show()

    '''Bagian Analitik'''
    X, Y = np.meshgrid(x_space, y_space)
    R = analytic(X, Y)
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    X, Y = np.meshgrid(x_space, y_space)
    Z = analytic(X, Y)
    surf = ax.plot_surface(X, Y, R, cmap=cm.coolwarm, linewidth=0, antialiased=False)
    ax.set_xlabel('$x$')
    ax.set_ylabel('$y$')
    plt.savefig('gambar-analitik-mse-case4-iterasi-{}.png'.format(a))
    plt.close()

    fig3 = plt.figure()
    plt.plot(epoch, loss)
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.grid(b=None, which='both', axis='both')
    plt.savefig('case4-mse-loss-epoch-{}.png'.format(a))
    plt.close()

    with open('RMSEcase4-mse-{}-epoch.csv'.format(a), 'w') as csvFile:
        writer = csv.writer(csvFile, delimiter=',', lineterminator='\n')
        delta_ANN = np.sqrt(mean_squared_error(R, surface2))
        delta_Numerik = np.sqrt(mean_squared_error(R, surface2))
        writer.writerow(['epoch', 'RMSE Ann Analitik', 'RMSE Numerik Analitik', 'loss'])
        writer.writerow(['{}'.format(a), delta_ANN, delta_Numerik, loss[a-1]])
    print 'nilai RMSE', delta_ANN
```

```
x = [50]
for a in x:
    main(a)
```

Repository Universitas Brawijaya