IF3270 Pembelajaran Mesin

Tugas Besar 2

Implementasi *Feedforward Neural Network (FFNN)* menggunakan Python

Disusun oleh:

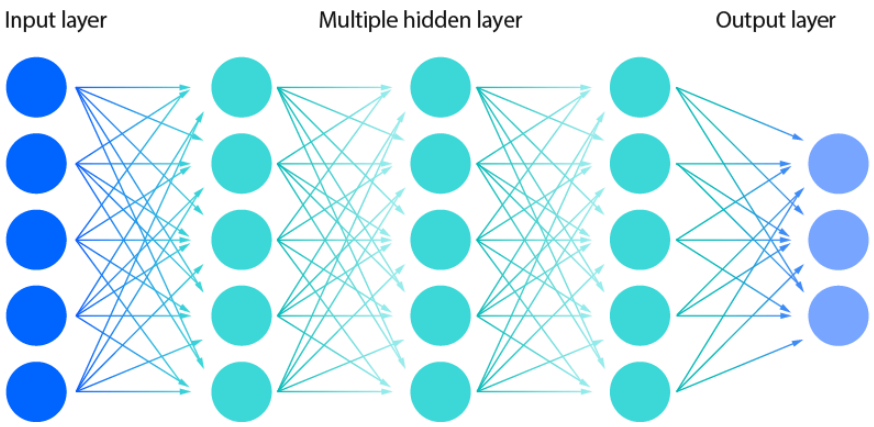
1. Zaki Yudhistira Candra 13522031
2. Edbert Eddyson G. 13522039
3. Vanson Kurnialim 13522049

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024/2025

Daftar Isi

Deskripsi Persoalan

.



Pada tugas besar ini, peserta diminta untuk membuat implementasi dari Feedforward Neural Network (FFNN) *from scratch* menggunakan bahasa Python. Mahasiswa diharapkan memahami konsep dasar neural network, termasuk forward propagation, backward propagation, dan pembaruan bobot (weight update).

Dalam implementasi ini, mahasiswa harus memastikan bahwa FFNN yang dibuat dapat:

- Menerima konfigurasi arsitektur berupa jumlah neuron per layer.
- Mendukung beberapa fungsi aktivasi, termasuk Linear, ReLU, Sigmoid, Tanh, dan Softmax.
- Menggunakan beberapa *loss function*, seperti Mean Squared Error (MSE), Binary Cross-Entropy, dan Categorical Cross-Entropy.
- Mendukung berbagai metode inisialisasi bobot, seperti zero *initialization*, distribusi uniform, dan distribusi normal.
- Melakukan forward propagation untuk menghitung output berdasarkan input dan parameter jaringan.
- Melakukan backward propagation untuk menghitung gradien dan memperbarui bobot menggunakan gradient descent.
- Mendukung pelatihan model dengan parameter batch size, learning rate, dan jumlah epoch, serta menampilkan histori loss selama pelatihan.
- Menyediakan mekanisme visualisasi untuk struktur jaringan, distribusi bobot, dan distribusi gradien dalam bentuk graf.

Selain itu, model yang telah dibentuk dapat diuji menggunakan dataset mnist_784 serta membandingkan hasilnya dengan implementasi yang ada di scikit-learn (MLPClassifier). Model juga *method* untuk simpan dan muat.

Pembahasan

1. Deskripsi Implementasi
    a. Berikut daftar file yang berisi kelas dan fungsi yang sudah diimplementasikan

Main.py Driver utama untuk menjalankan program.	
activation_functions_dict	Dictionary berisi fungsi aktivasi
savePickle()	Fungsi untuk menyimpan data pelatihan (data_train) dan labelnya (data_train_class) ke file pickle.
loadPickle()	Fungsi untuk memuat data pelatihan (data_train) dan labelnya (data_train_class) dari file pickle.
start_program()	Fungsi untuk memulai program dengan memberikan pilihan kepada pengguna untuk membuat konfigurasi jaringan Neural Network baru, memuat konfigurasi dari file JSON, atau memuat konfigurasi yang sudah ada.
initiateEngine()	Fungsi untuk menginisialisasi mesin (engine) berdasarkan konfigurasi yang diberikan.

DataLoader.py Modul untuk memuat dataset dari file ARFF dan memisahkan fitur dengan label kelas.	
Class DataLoader	Kelas untuk memuat dataset dari file ARFF dan mengonversinya ke format numpy array.
__init__(self, path)	Konstruktor untuk inisialisasi objek DataLoader dengan path file ARFF.
load_data(self)	Metode untuk memuat dataset, memisahkan fitur dan label kelas, serta mencatat waktu pemrosesan.

Engine.py Modul berisi Layer, NeuralNetwork, dan Engine

Class Layer	Kelas untuk merepresentasikan lapisan (layer) dalam jaringan saraf tiruan.
__init__(self, path)	Konstruktor untuk inisialisasi parameter lapisan seperti jumlah neuron, fungsi aktivasi, bias, dan metode inisialisasi bobot.
multiply(self, input_array)	Metode untuk melakukan perkalian matriks antara input dan bobot lapisan, serta menerapkan fungsi aktivasi.
update_weights(self, ...)	Metode untuk memperbarui bobot menggunakan algoritma backpropagation dengan gradient descent.
initiateWeightRDUniform(...)	Metode untuk memperbarui bobot menggunakan algoritma backpropagation dengan gradient descent.
initiateWeightRDUniform(...)	Inisialisasi bobot secara acak seragam (uniform random) dalam rentang tertentu.
initiateWeightRDNormal(...)	Inisialisasi bobot secara acak normal (normal random) dengan mean dan variansi tertentu.
initiateWeightZero()	Inisialisasi bobot menjadi nol.
initiateWeightXavier()	Inisialisasi bobot menggunakan metode Xavier untuk distribusi bobot yang sesuai dengan ukuran lapisan.
initateWeightHe()	Inisialisasi bobot menggunakan metode He untuk distribusi bobot yang sesuai dengan ukuran lapisan (biasanya untuk fungsi aktivasi ReLU).
class NeuralNetwork	Kelas yang mengelola seluruh lapisan dalam jaringan saraf, termasuk forward pass dan backpropagation.
__init__(self, ...)	Konstruktor untuk inisialisasi parameter jaringan saraf seperti jumlah input, output, lapisan tersembunyi, dan fungsi aktivasi
initiateLayers(self, bias)	Metode untuk menginisialisasi semua lapisan dalam jaringan saraf berdasarkan parameter yang diberikan
forward(self, input_array)	Metode untuk melakukan forward pass melalui semua lapisan dalam jaringan saraf.
backward(self, ...)	Metode untuk melakukan backpropagation, menghitung delta error, dan memperbarui bobot.
train(self, inputs, ...)	Metode untuk melatih jaringan saraf menggunakan

	algoritma backpropagation untuk sejumlah epoch.
class Engine	Kelas yang bertanggung jawab untuk mengelola data pelatihan, konfigurasi jaringan saraf, dan proses pelatihan.
__init__(self, ...)	Konstruktor untuk inisialisasi parameter engine, termasuk data pelatihan, fungsi aktivasi, dan parameter lainnya.
batch_train(self)	Metode untuk melakukan pelatihan dalam batch kecil.
train_backprop(self)	Metode untuk melatih jaringan saraf menggunakan algoritma backpropagation.
saveANNasPickle(self, name)	Metode untuk melatih jaringan saraf menggunakan algoritma backpropagation.
loadANNfromPickle(name)	Metode statik untuk memuat objek jaringan saraf dari file pickle.

classes.py Modul untuk mengelola konfigurasi jaringan saraf tiruan, termasuk menyimpan dan memuat konfigurasi dari file JSON.	
class Configuration	Kelas untuk merepresentasikan konfigurasi jaringan saraf tiruan.
__init__(self, path)	Konstruktor untuk inisialisasi parameter konfigurasi seperti nama konfigurasi, batch size, learning rate, jumlah epoch, fungsi aktivasi, dll.
to_dict(self)	Metode untuk mengonversi objek Configuration menjadi dictionary Python, sehingga dapat diserialisasi ke JSON
saveConfigtoJSON(config, ...)	Metode statik untuk menyimpan konfigurasi ke file JSON.
loadConfigfromJSON(filepath)	Metode statik untuk memuat konfigurasi dari file JSON dan mengembalikan objek Configuration
class Neuron	Kelas sederhana untuk merepresentasikan neuron dengan nilai tertentu.
__init__(self, value=0)	Konstruktor untuk inisialisasi nilai neuron (default: 0).
Attribute value	Nilai dari neuron

class Weight	Kelas untuk merepresentasikan bobot dengan jenis tertentu dan parameter terkait.
__init__(self, weight_type, ...)	Konstruktor untuk inisialisasi jenis bobot dan parameter bobot.
Attribute Type	Tipe weight
Attribute Parameter	Parameter masukan

bonus.py Modul untuk menyediakan metode inisialisasi bobot seperti Xavier dan He..	
class InitializationMethode	Kelas yang menyediakan metode inisialisasi bobot untuk jaringan saraf tiruan.
xavier_initialization(...)	Metode statik untuk menginisialisasi bobot menggunakan teknik Xavier (Glorot).
he_initialization(...)	Metode statik untuk menginisialisasi bobot menggunakan teknik He.
class Normalization	Kelas yang menyediakan metode normalisasi data.
RMSnorm(self, dim, eps=1e-6)	Metode untuk menerapkan RMS Normalization pada data dengan dimensi tertentu.

load.py File untuk memuat konfigurasi jaringan saraf tiruan dari file JSON.	
load_init()	Fungsi untuk memuat konfigurasi jaringan saraf tiruan dari file JSON.

LossFunction.py Modul untuk menyediakan fungsi-fungsi loss (fungsi kerugian) yang umum digunakan dalam jaringan saraf tiruan.	
class LossLib	Kelas yang menyediakan implementasi fungsi loss seperti Mean Squared Error (MSE), Binary Cross-Entropy (BCE), dan Categorical Cross-Entropy (CCE).
mean_squared_error(...)	Metode untuk menghitung Mean Squared Error (MSE) antara nilai target (y_true) dan prediksi (y_pred).

binary_cross_entropy(...)	Metode untuk menghitung Binary Cross-Entropy (BCE) antara nilai target (y_true) dan prediksi (y_pred).
categorical_cross_entropy(...)	Metode untuk menghitung Categorical Cross-Entropy (CCE) antara nilai target (y_true) dan prediksi (y_pred).

Misc.py Modul tambahan	
activation_functions	Dictionary yang menyimpan fungsi aktivasi seperti ReLU, Sigmoid, Tanh, Linear, dan Softmax.
getPositiveInteger(prompt)	Fungsi untuk mendapatkan input integer positif dari pengguna.
getPositiveFLoat(prompt)	Fungsi untuk mendapatkan input float positif dari pengguna.
getLossFunction()	Fungsi untuk meminta pengguna memilih fungsi loss (MSE, Cross-Entropy, atau Categorical Cross-Entropy).
getHiddenFunction()	Fungsi untuk meminta pengguna memilih fungsi aktivasi untuk lapisan tersembunyi.
getInitType()	Fungsi untuk meminta pengguna memilih metode inisialisasi bobot (Random Uniform, Random Normal, He, Xavier, Zeros).

b. Forward Propagation

```
1 def forward(self, input_array):
2     for layer in self.layers:
3         input_array = np.append(input_array, 1)
4         input_array = layer.multiply(input_array)
5     return input_array
```

Data pada input layer akan dibentuk menjadi sebuah matrix. Selanjutnya akan dilakukan iterasi untuk setiap layer. Pada awal tiap layer setiap baris pada matrix akan ditambahkan nilai 1 sebagai value dari bias lalu dilakukan perkalian dot product antara matrix data dengan matrix bobot. Hasil yang didapatkan akan dimasukkan ke dalam fungsi aktivasi layer tersebut dan hasilnya akan menjadi output pada layer tersebut. Pada akhir iterasi akan didapatkan nilai hasil pada layer output terakhir.

Fungsi forward pada gambar di atas memiliki dua parameter yaitu self dan input_array. Fungsi ini merupakan fungsi dari kelas NeuralNetwork sehingga menggunakan self, fungsi dapat mengakses banyak hal. Input_array merupakan data matriks yang akan di-*forward*. Fungsi ini akan dipanggil pada setiap iterasi layer.

```
def multiply(self, input_array):
        ret = np.matmul(input_array, self.weight_matrix)
        ret = ret.astype(np.float64)
        self.output = self.activation_function(ret)
        return self.output
```

Fungsi multiply melakukan operasi dot product dan juga memanggil fungsi aktivasi yang disimpan di dalam tiap layer. Fungsi ini dipanggil oleh fungsi forward.

c. Backward Propagation

```
def backward(self, input_array, forwarded_result, expected_output, learning_rate):
        last_layer = self.layers[-1]
        # Calculate delta for the output layer
        last_layer.delta = (forwarded_result - expected_output) *
self.output_layer_function_derivative(last_layer.output)

        # Backpropagate through the hidden layers
        for i in reversed(range(len(self.layers) - 1)):
            layer = self.layers[i]
            next_layer = self.layers[i + 1]

            # Calculate the error term for the current layer
            error_term = np.dot(next_layer.delta, next_layer.weight_matrix[:-1, :].T)
            layer.delta = error_term * layer.activation_function_derivative(layer.output)

        batch_size = input_array.shape[0]
        for i, layer in enumerate(self.layers):
            if i == 0:
                input_with_bias = np.hstack((input_array, np.ones((batch_size, 1))))
            else:
                prev_output = self.layers[i - 1].output
                input_with_bias = np.hstack((prev_output, np.ones((batch_size, 1))))

            layer.updateWeights(input_with_bias, learning_rate)
```

Fungsi backward ini merupakan fungsi yang mengurusi proses backward propagation. Dimulai dari output layer atau layer terakhir, akan dihitung delta sesuai dengan rumusnya. Pada perhitungan delta, digunakan turunan dari fungsi aktivasi yang didapatkan dari informasi pada tiap layer.

Mulai dari layer terakhir, akan dihitung delta tiap layer sampai layer pertama. Dalam prosesnya, akan dihitung nilai error atau error_term. Ketika semua proses sudah selesai, maka semua data yang diperlukan sudah tersedia untuk melakukan perbaruan bobot untuk tiap neuron pada tiap layer.

d. Weigh Update

```
def updateWeights(self, input_array, learning_rate):
    self.delta = self.delta.astype(np.float64)
    input_array = input_array.astype(np.float64)
    gradient = input_array.T @ self.delta / input_array.shape[0]
    self.weight_matrix -= learning_rate * gradient
```

Fungsi updateWeights menerima input berupa learning rate dan matriks data. Berdasarkan parameter tersebut akan dihitung nilai gradien descent dengan rumus :

$$\nabla W = \frac{1}{m} X^T \cdot \delta$$

- XT: transpos dari matriks data
- δ: delta (error yang dibackpropagate), didapatkan pada proses sebelumnya
- m: jumlah data dalam batch

Gradien ini akan dikalikan dengan learning rate untuk mendapatkan nilai bobot baru. Fungsi ini dipanggil untuk tiap layer untuk melakukan update pada bobot tiap neuron dan juga bias.

2. Hasil Pengujian

Berikut adalah bab pengujian model, terdapat beberapa sub bab pada bab ini yang berisi variabel berbeda dan pengaruhnya terhadap model yang dibuat. Setiap pengujian di sini akan menggunakan *batch size* sebesar 32.

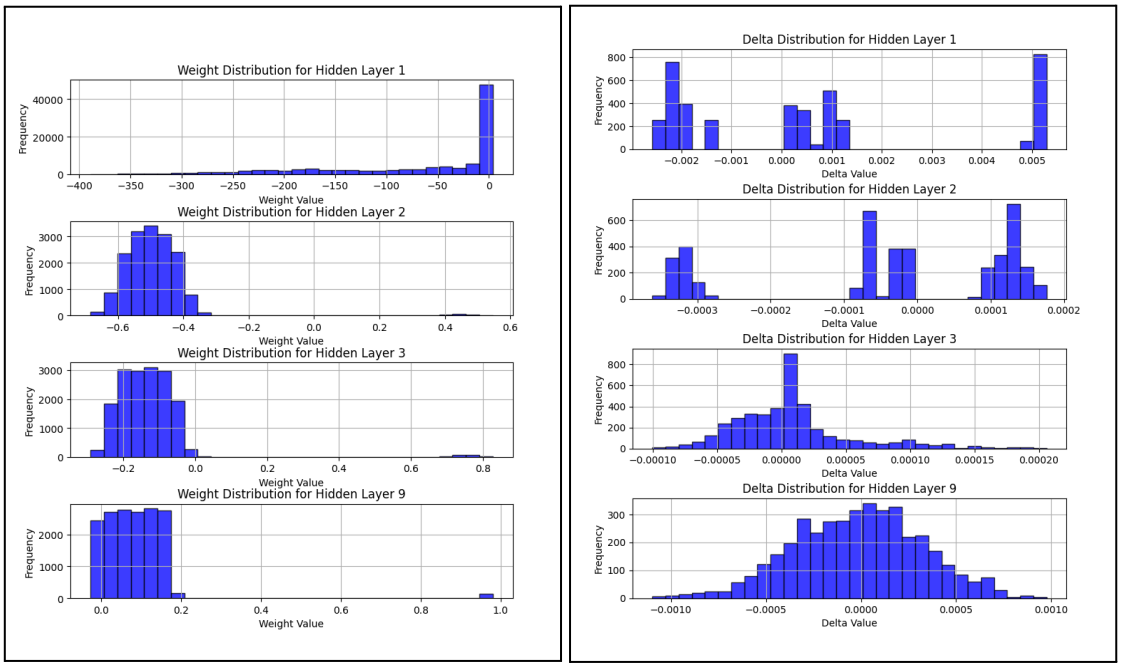
a. Pengaruh depth dan width

Akan dilakukan pengujian terhadap *depth* dan *width* dan performanya terhadap *neural network* yang dibentuk.

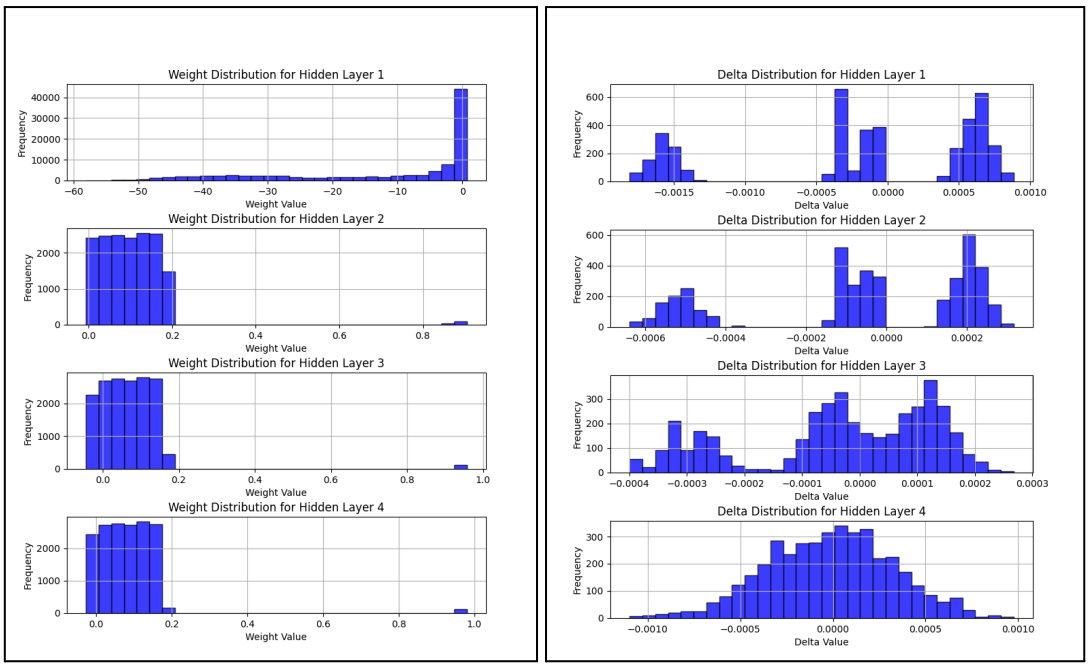
1. Pengujian *depth*:
    a. Semua *layer* menggunakan *sigmoid*
    b. Digunakan *mse* sebagai perhitungan error
    c. Setiap *layer* akan berisi 128 simpul
    d. Digunakan *learning rate* sebesar 0.01
    e. Digunakan *random gaussian* dengan 0 sebagai min dan 0.2 sebagai nilai maksimum

Tabel 2.1. Pengaruh depth terhadap model

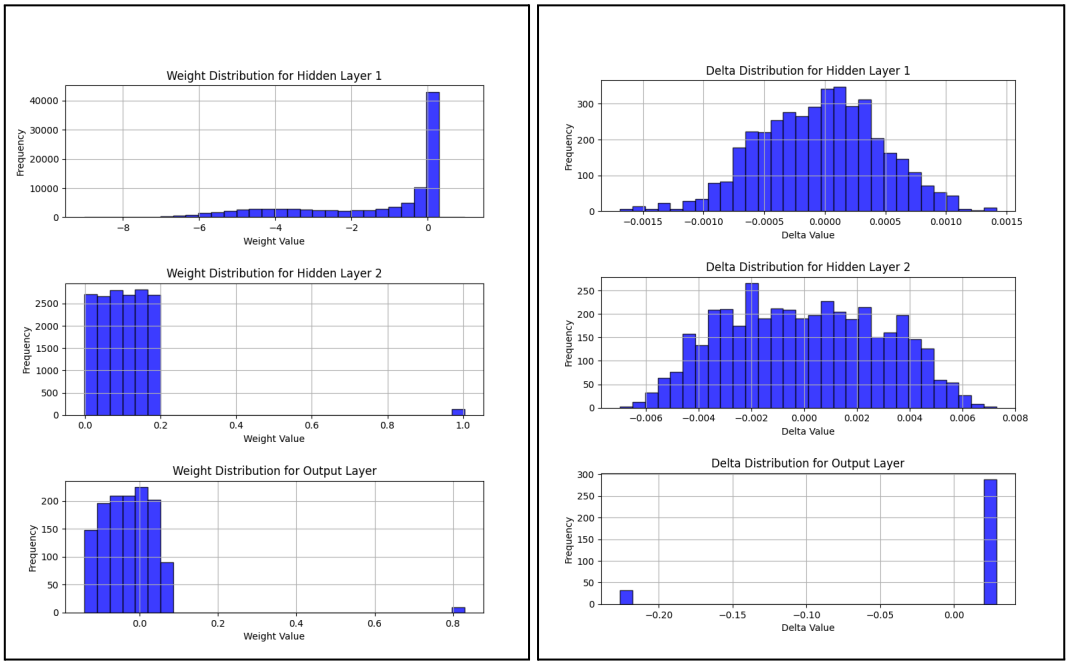
Depth (Jumlah simpul)	Waktu (Rata-rata per *epoch*)	Akurasi (Menggunakan *MSE*)
10	~145 detik	0.08999135513365199
5	~132 detik	0.08999135513365286
3	~137 detik	0.08997942336441864



Gambar 2.1 Distribusi bobot dan delta untuk depth=10 note tidak semua distribusi layer diperlihatkan, hanya untuk layer 0, 1, 2, 8*



Gambar 2.2 Distribusi bobot dan delta untuk depth=5 note tidak semua distribusi layer diperlihatkan, hanya untuk layer 0, 1, 2, 3*



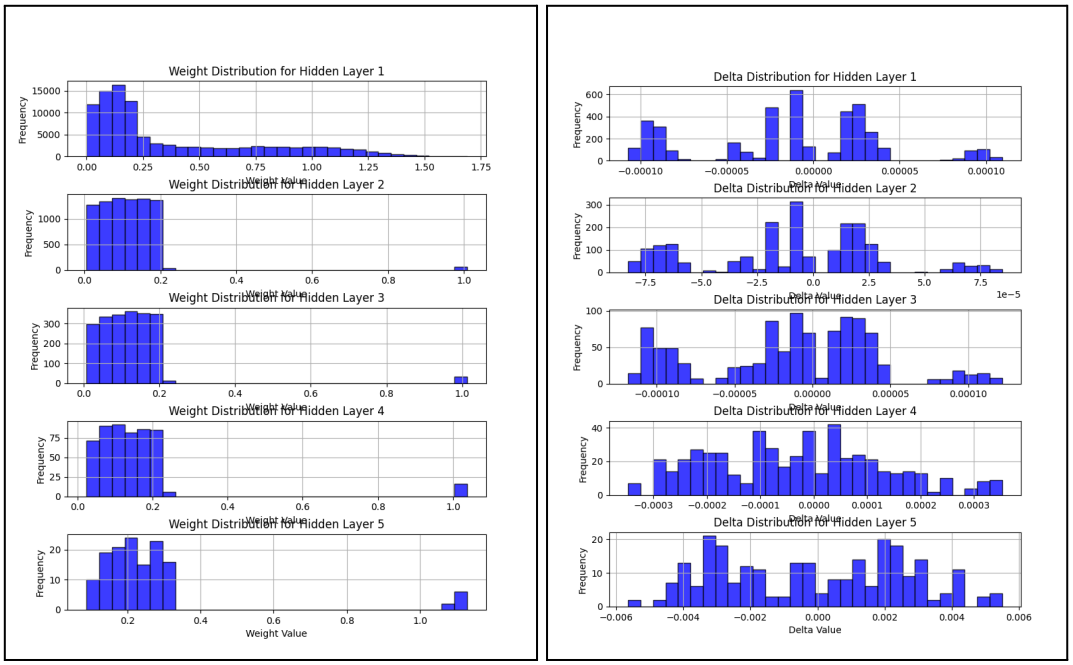
Gambar 2.3 Distribusi bobot dan delta untuk depth=2 note

Dari pengujian ini, terlihat tidak begitu ada perbedaan yang signifikan antara kedalaman model.

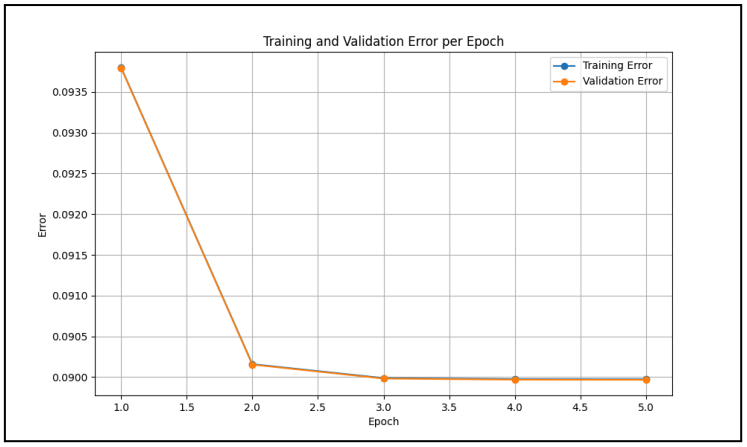
2. Pengujian *width*
    a. Semua *layer* menggunakan *sigmoid*
    b. Digunakan *mse* sebagai perhitungan error
    c. Setiap *model* akan memiliki kedalaman 5 simpul
    d. Digunakan *learning rate* sebesar 0.01
    e. Digunakan *random gaussian* dengan 0 sebagai min dan 0.2 sebagai nilai maksimum

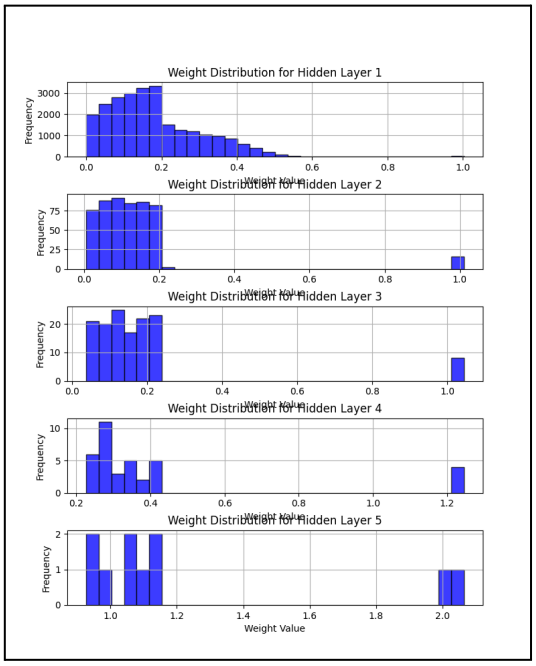
Tabel 2.2. Pengaruh width terhadap model

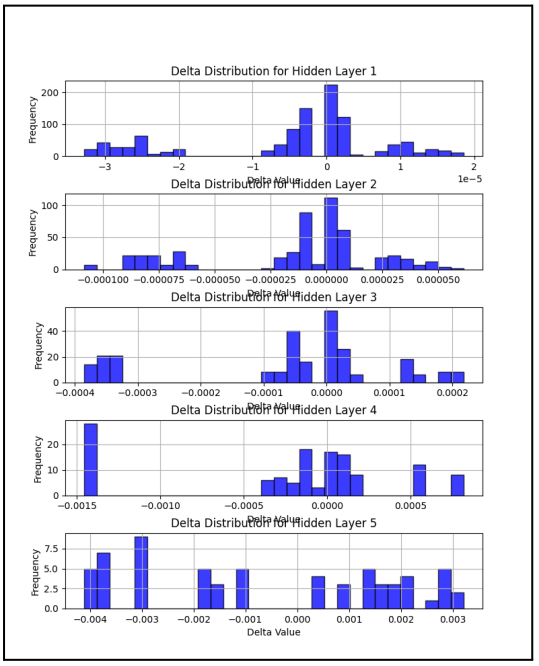
Urutan Width	Waktu (Rata-rata per *epoch*)	Akurasi (Menggunakan *MSE*)
128,64,32,16,8	~143 detik	0.08996687077765304
32,16,8,4,2	~37 detik	0.08996613185966737
2,4,8	~137 detik	0.08996972493741628



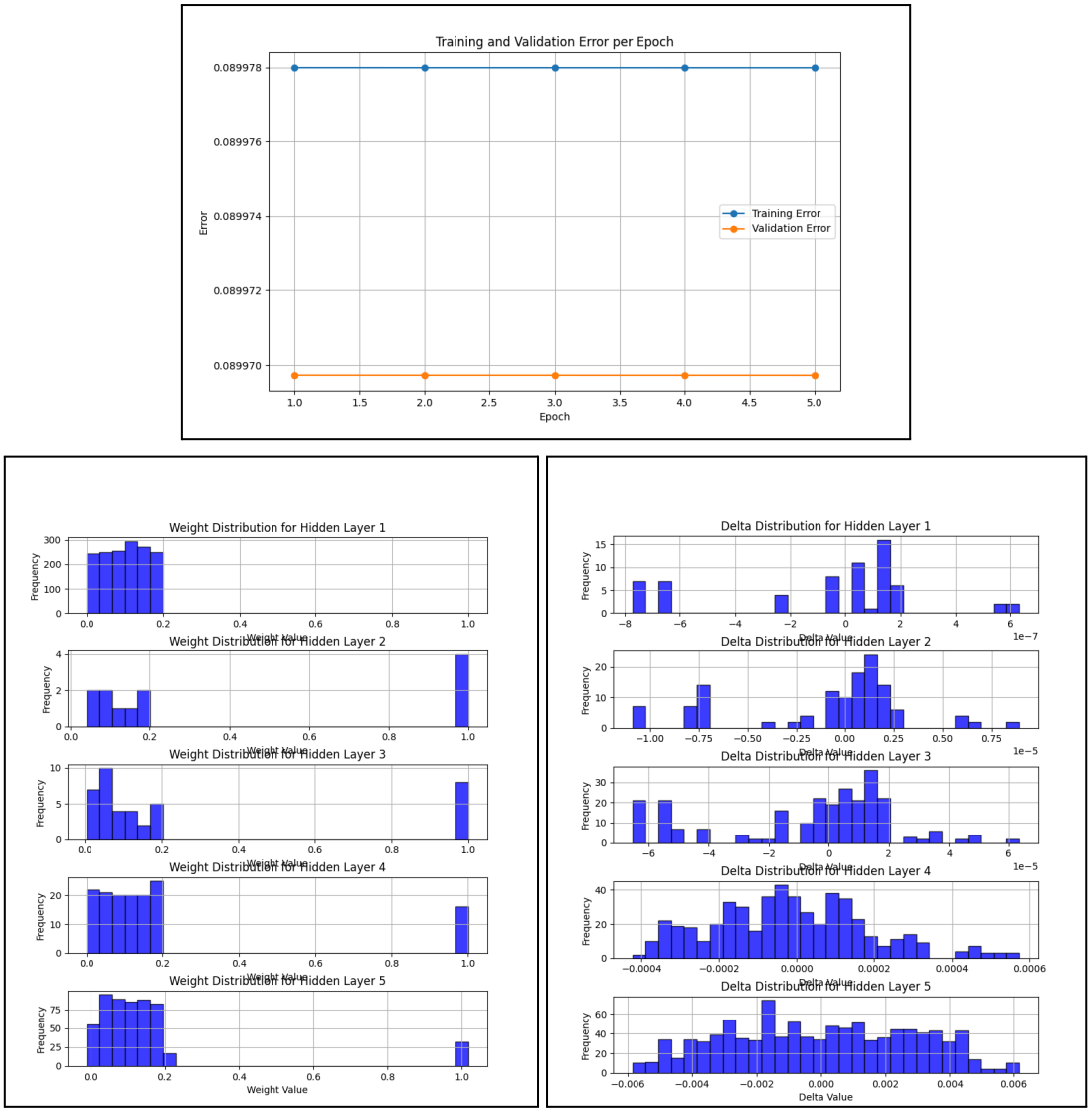
Gambar 2.4 Distribusi bobot dan delta untuk varian width note tidak semua distribusi layer diperlihatkan, hanya untuk layer 0, 1, 2, 3, 4*







Gambar 2.5 Distribusi bobot dan delta untuk varian width note tidak semua distribusi layer diperlihatkan, hanya untuk layer 0, 1, 2, 3, 4*



Gambar 2.5 Distribusi bobot dan delta untuk varian width note serta plot grafik validasi error dan training tidak semua distribusi layer diperlihatkan, hanya untuk layer 0, 1, 2, 3, 4*

b. Pengaruh fungsi aktivasi

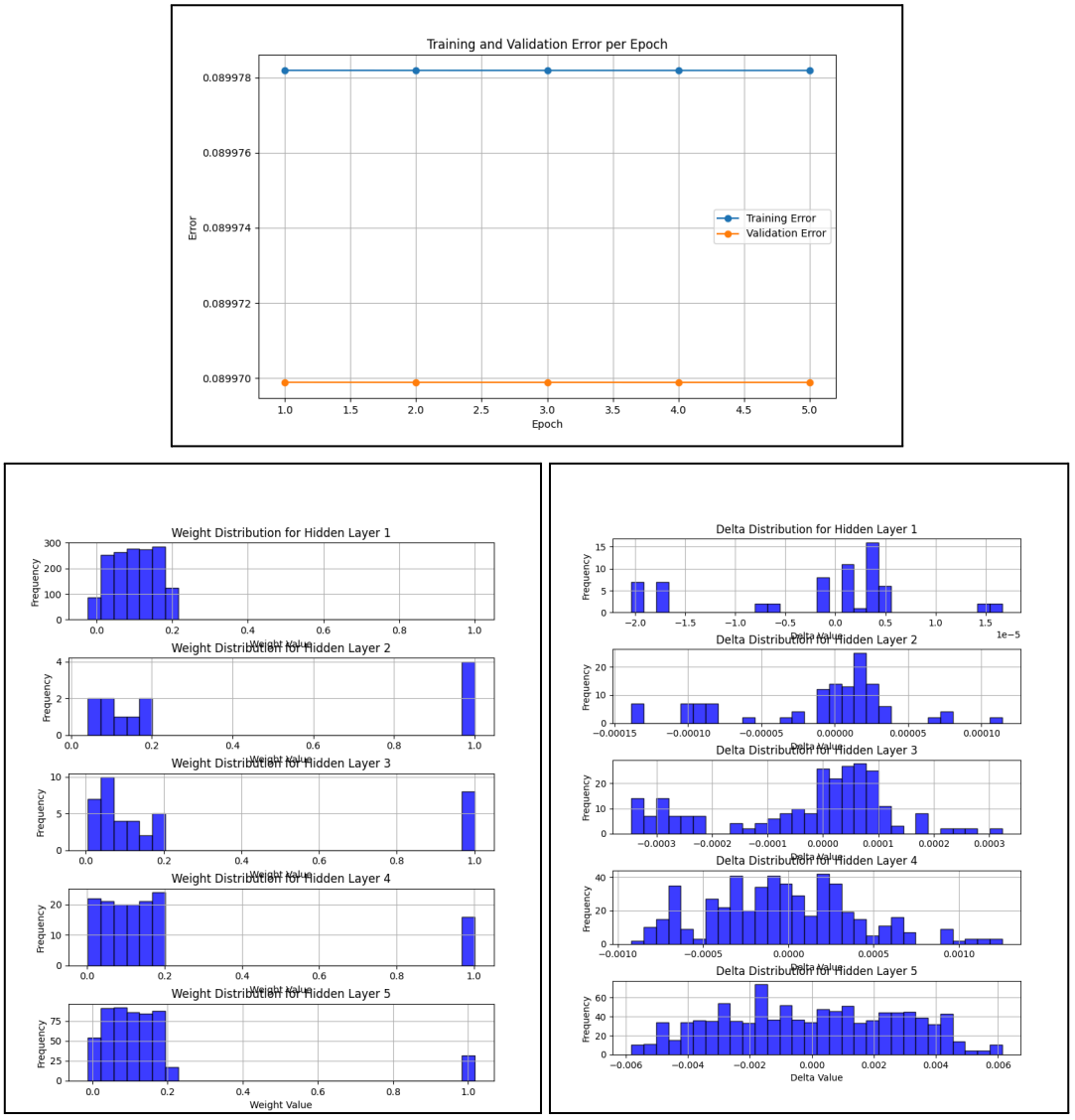
Akan dilakukan pengujian terhadap *activation function* yang berbeda pada tiap *layer*, berikut adalah detail spesifikasi pengujian.

1. Memiliki kedalaman 5 *layer*
2. Digunakan *mse* sebagai perhitungan error
3. Setiap *layer* akan berisi 2, 4, 8, 16, 32 simpul
4. Digunakan *learning rate* sebesar 0.01
5. Digunakan *random gaussian* dengan 0 sebagai min dan 0.2 sebagai nilai maksimum

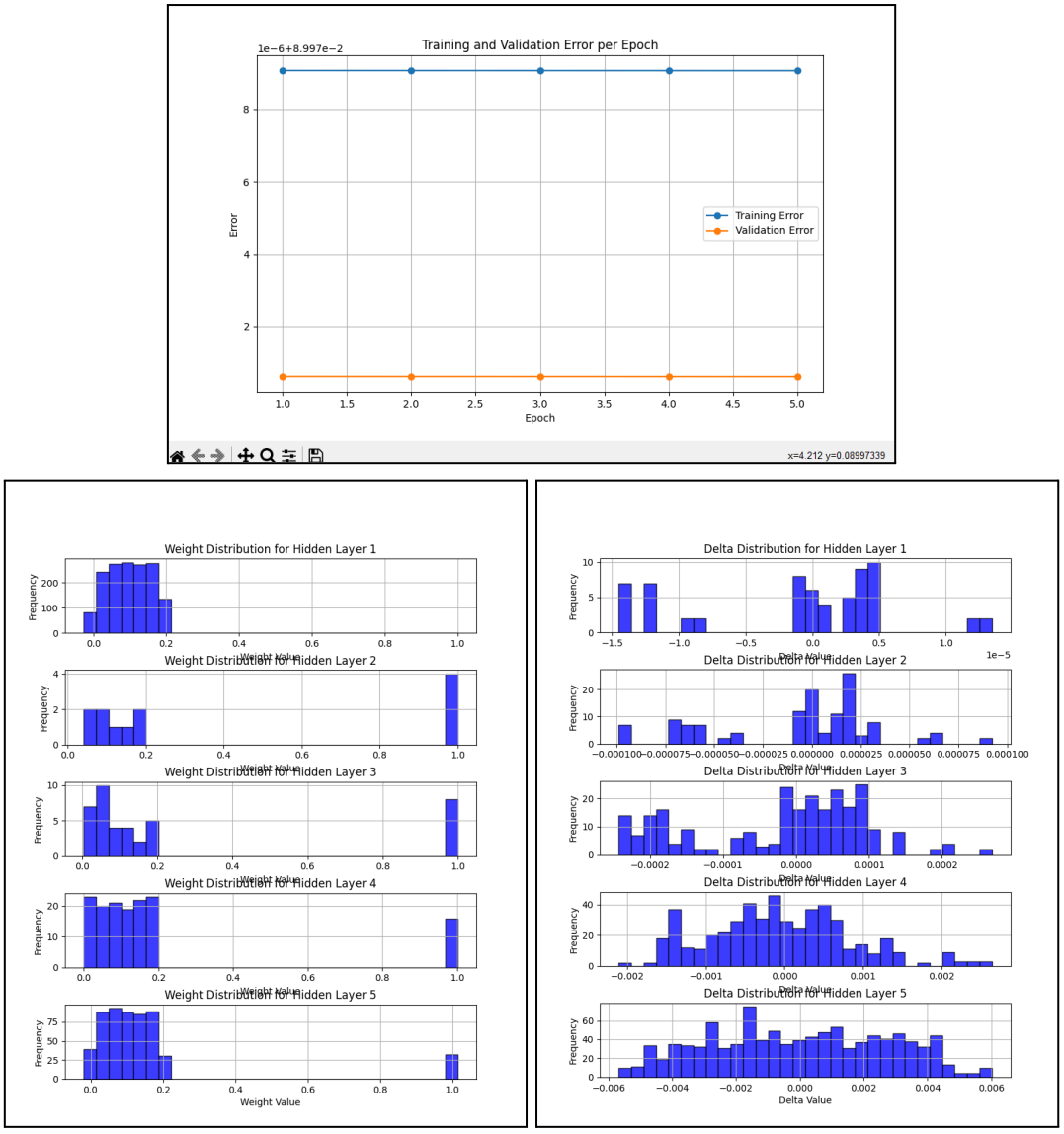
Tabel 2.3. Pengaruh activation function

Susunan *activation function*	Waktu (Rata-rata per *epoch*)	Akurasi (Menggunakan *MSE*)
"tanh","tanh"si	~34	0.0899706095800385

gmoid","linear","sigmoid"		
"tanh","tanh","tanh","tanh","sigmoid"	~2 detik	0.08999135513365286
3	~137 detik	0.08997942336441864



Gambar 2.7 Distribusi bobot dan delta untuk varian activation function note tidak semua distribusi layer diperlihatkan, hanya untuk layer 0, 1, 2, 3, 4*



Gambar 2.8 Distribusi bobot dan delta untuk varian activation function note tidak semua distribusi layer diperlihatkan, hanya untuk layer 0, 1, 2, 3, 4*

c. Pengaruh learning rate {Tidak sempat}
d. Pengaruh inisialisasi bobot {Tidak sempat}
e. Perbandingan dengan library sklearn

```
Loading MNIST dataset...
Training MLPClassifier (ANN)...
/usr/local/lib/python3.11/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  warnings.warn(

MSE accuracy MLPClassifier: 0.090010
```

Akurasi model ANN yang dibuat : 08999135513365199
Akurasi model Sklearn : 090010

Model Sklearn yang digunakan memiliki konfigurasi yang sama dengan model yang digunakan untuk pengujian bagian (a) yang menggunakan

10 depth. Kedua hasil pengujian memiliki perbedaan yang sangat kecil sehingga bisa dikatakan model ANN yang kami buat cukup baik karena dapat menyamai hasil dari library Sklearn. Apalagi pada kasus ini, akurasi model kami sedikit lebih baik karena lebih mendekati 0.

Walau hasil yang didapatkan dari model buatan kami sudah baik, namun waktu eksekusi dalam proses training dan pengujian masih terpantau cukup lama jika dibandingkan dengan waktu pada model Sklearn. Hal ini terjadi karena model kami belum dapat dioptimasi secara efektif seperti library Sklearn sehingga proses dapat berjalan dengan sangat efisien.

Kesimpulan dan Saran

Pembelajaran dengan cara membangun model FFNN *from scratch* merupakan cara yang baik untuk meningkatkan pemahaman terhadap konsep ANN. Hanya saja, pada implementasi ini, karena satu dua lain hal, kelompok kami tidak dapat menyelesaikannya secara maksimal, meski sudah diberikan perpanjangan *deadline* dan waktu yang cukup lama. Meski spesifikasi sudah cukup terpenuhi, beberapa *bug* yang muncul saat *testing* membuat program tidak dapat diuji secara maksimal. Kedepannya, hal ini merupakan refleksi yang sangat berarti bagi kami dan mencoba untuk terus berkembang kedepannya.

Pembagian Tugas

No	NIM	Tugas
1	13522031	Menyusun laporan, Mengimplementasi ANN
2	13522039	Menyusun laporan, Mengimplementasi tampilan ANN
3	13522049	Menyusun laporan, Mengimplementasi simpan dan muat ANN

Referensi

Spesifikasi Tugas: Spesifikasi Tugas Besar 1 IF3270 Pembelajaran Mesin
Salindia Kuliah Pembelajaran Mesin FFNN: Salindia FFNN
Dataset MNIST: https://www.openml.org/d/554