

LAPORAN TUGAS BESAR 01

IF3270 Pembelajaran Mesin

Feedforward Neural Network



Disusun Oleh:

Juan Alfred Widjaya 13522073

Albert 13522081

Ivan Hendrawan Tan 13522111

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

DAFTAR ISI	1
DAFTAR GAMBAR	2
DAFTAR TABEL	4
BAB I DESKRIPSI PERMASALAHAN	5
BAB II PEMBAHASAN	7
A. Penjelasan Implementasi	7
a. Deskripsi Kelas beserta Deskripsi Atribut dan Methodnya	7
b. Penjelasan Forward Propagation	14
c. Penjelasan Backward Propagation dan Weight Update	15
B. Hasil Pengujian	16
a. Pengaruh Depth dan Width	16
b. Pengaruh Fungsi Aktivasi	18
c. Pengaruh Learning Rate	21
d. Pengaruh Inisialisasi Bobot	22
e. Pengaruh Regularisasi	25
f. Pengaruh Normalisasi RMSNorm	28
g. Perbandingan dengan Library Sklearn	30
BAB III KESIMPULAN DAN SARAN	31
A. Kesimpulan	31
B. Saran	31
PEMBAGIAN TUGAS	32
REFERENSI	33

DAFTAR GAMBAR

Gambar 1.1 Ilustrasi Feedforward Neural Network dengan Multiple Layer	6
Gambar 2.2.1.1 Kurva Loss Variasi Depth	17
Gambar 2.2.1.2 Kurva Loss Variasi Width	18
Gambar 2.2.2.1 Kurva Loss Variasi Fungsi Aktivasi	19
Gambar 2.2.2.2 Distribusi Bobot Sigmoid pada Layer-1	19
Gambar 2.2.2.3 Distribusi Gradien Sigmoid pada Layer-1	19
Gambar 2.2.2.4 Distribusi Bobot Linear pada Layer-1	20
Gambar 2.2.2.5 Distribusi Gradien Linear pada Layer-1	20
Gambar 2.2.2.6 Distribusi Bobot Elu pada Layer-1	20
Gambar 2.2.2.7 Distribusi Gradien Elu pada Layer-1	20
Gambar 2.2.2.8 Distribusi Bobot Tanh pada Layer-1	20
Gambar 2.2.2.9 Distribusi Gradien Tanh pada Layer-1	21
Gambar 2.2.2.10 Distribusi Bobot LeakyReLU pada Layer-1	21
Gambar 2.2.2.11 Distribusi Gradien LeakyReLU pada Layer-1	21
Gambar 2.2.2.12 Distribusi Bobot ReLU pada Layer-1	21
Gambar 2.2.2.13 Distribusi Gradien ReLU pada Layer-1	21
Gambar 2.2.3.1 Kurva Loss Variasi Learning Rate	22
Gambar 2.2.4.1 Kurva Loss Variasi Inisialisasi Bobot	23
Gambar 2.2.4.2 Distribusi Bobot Inisialisasi Zero pada Layer-1	24
Gambar 2.2.4.3 Distribusi Gradien Inisialisasi Zero pada Layer-1	24
Gambar 2.2.4.4 Distribusi Bobot Inisialisasi Random_normal pada Layer-1	24
Gambar 2.2.4.5 Distribusi Gradien Inisialisasi Random_normal pada Layer-1	24
Gambar 2.2.4.6 Distribusi Bobot Inisialisasi Random_uniform pada Layer-1	24
Gambar 2.2.4.7 Distribusi Gradien Inisialisasi Random_uniform pada Layer-1	25
Gambar 2.2.4.8 Distribusi Bobot Inisialisasi Xavier pada Layer-1	25
Gambar 2.2.4.9 Distribusi Gradien Inisialisasi Xavier pada Layer-1	25
Gambar 2.2.4.10 Distribusi Bobot Inisialisasi He pada Layer-1	25
Gambar 2.2.4.11 Distribusi Gradien Inisialisasi He pada Layer-1	25
Gambar 2.2.5.1 Kurva Loss Variasi Regularisasi	26
Gambar 2.2.5.2 Distribusi Bobot Regularisasi L1 + L2	27
Gambar 2.2.5.3 Distribusi Gradien Regularisasi L1 + L2	27
Gambar 2.2.5.4 Distribusi Bobot Regularisasi L2	27
Gambar 2.2.5.5 Distribusi Gradien Regularisasi L2	27
Gambar 2.2.5.6 Distribusi Bobot Regularisasi L1	27
Gambar 2.2.5.7 Distribusi Gradien Regularisasi L1	28
Gambar 2.2.5.8 Distribusi Bobot Tanpa Regularisasi	28
Gambar 2.2.5.9 Distribusi Gradien Tanpa Regularisasi	28
Gambar 2.2.6.1 Kurva Loss Variasi Normalisasi dengan RMSNorm	29
Gambar 2.2.6.2 Distribusi Bobot Tanpa Normalisasi	29
Gambar 2.2.6.3 Distribusi Gradien Tanpa Normalisasi	29
Gambar 2.2.6.4 Distribusi Bobot dengan Normalisasi RMSNorm	29

Gambar 2.2.6.5 Distribusi Gradien dengan Normalisasi RMSNorm	30
Gambar 2.2.7.1 Perbandingan Hasil Model Buatan dengan MLP Sklearn	30

DAFTAR TABEL

Tabel 2.2.1.1 Akurasi Variasi Depth	17
Tabel 2.2.1.2 Akurasi Variasi Width	17
Tabel 2.2.2.1 Akurasi Variasi Fungsi Aktivasi	18
Tabel 2.2.3.1 Akurasi Variasi Learning Rate	22
Tabel 2.2.4.1 Akurasi Variasi Inisialisasi Bobot	23
Tabel 2.2.5.1 Akurasi Variasi Regularisasi	26
Tabel 2.2.6.1 Akurasi Variasi Normalisasi RMSNorm	28

BAB I

DESKRIPSI PERMASALAHAN

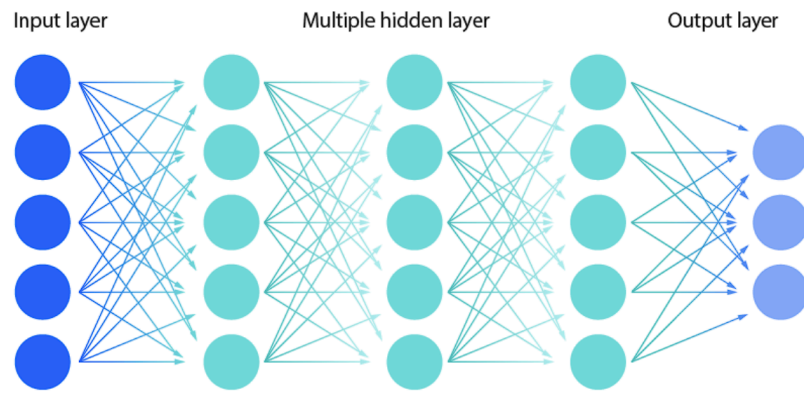
Feedforward Neural Network (FFNN) adalah salah satu jenis jaringan saraf buatan (artificial neural network) di mana informasi mengalir secara satu arah sehingga membentuk directed acyclic graph (DAG) dengan node input dan node output yang sudah ditentukan. Proses bermula dari satu atau beberapa input layer, ke satu atau beberapa hidden layer, dan berakhir di satu atau beberapa output layer tanpa adanya umpan balik (feedback loop).

Dalam FFNN, setiap neuron hanya menerima sinyal dari neuron pada lapisan sebelumnya dan mengirimkan sinyal ke neuron pada lapisan berikutnya sehingga prosesnya bersifat maju. Model ini merupakan dasar dari banyak algoritma pembelajaran mesin dan deep learning, model ini juga digunakan untuk mempelajari hubungan non-linear antara input dan output.

Modul FFNN yang dikembangkan harus menangani konfigurasi dalam menentukan jumlah neuron pada setiap layer dari input, hidden, hingga output serta berbagai fungsi aktivasi seperti Linear, ReLU, Sigmoid, tanh, dan Softmax. Selain itu, modul juga harus mendukung berbagai fungsi loss seperti Mean Squared Error (MSE), Binary Cross-Entropy, dan Categorical Cross-Entropy serta menyediakan opsi inisialisasi bobot dengan metode nol, random uniform, dan random normal, lengkap dengan parameter seperti batas minimal, batas maksimal, mean, variance, dan seed untuk reproduksibilitas.

Modul harus bisa menyimpan bobot dan gradien tiap neuron termasuk bias, menyediakan metode untuk menampilkan struktur jaringan beserta distribusi bobot dan gradien per layer serta fitur save dan load model. Proses forward propagation harus bisa menerima input berupa batch, sedangkan backward propagation diimplementasikan dengan menggunakan konsep chain rule untuk menghitung gradien yang kemudian digunakan dalam pembaruan bobot melalui gradient descent.

Selain itu, modul juga mencakup pengujian model dengan menganalisis pengaruh hyperparameter seperti depth, width, fungsi aktivasi, learning rate, dan metode inisialisasi, serta membandingkan hasil prediksi dengan model MLP yang dihasilkan dari library sklearn.



Gambar 1.1 Ilustrasi Feedforward Neural Network dengan Multiple Layer

BAB II

PEMBAHASAN

A. Penjelasan Implementasi

a. Deskripsi Kelas beserta Deskripsi Atribut dan Methodnya

Pada tugas besar ini, terdapat beberapa file yang digunakan untuk menerapkan Feedforward Neural Network seperti `activations.py`, `initializers.py`, `losses.py`, `main.ipynb`, `model.py`, `norm.py`, dan `utils.py`. Berikut adalah kelas-kelas yang terdapat pada setiap file.

1. `activations.py`

`ActivationFunction`, kelas ini mendefinisikan kerangka untuk fungsi aktivasi. Kelas ini memiliki dua method abstrak yaitu `forward(z)` dan `backward(z)`, `forward(z)` berfungsi untuk menghitung output fungsi aktivasi berdasarkan input z dan `backward(z)` berfungsi untuk menghitung turunan (gradien) dari fungsi aktivasi terhadap input z untuk proses backpropagation.

`LinearActivation` merupakan implementasi fungsi aktivasi linear. Kelas ini memiliki dua method yaitu `forward(z)` dan `backward(z)`. `forward(z)` mengembalikan nilai z tanpa ada perubahan (output sama dengan input) dan `backward(z)` mengembalikan array yang berisi angka 1 dengan ukuran yang sama seperti z (turunan dari fungsi linear adalah 1).

`ReLUActivation` mengimplementasikan fungsi Rectified Linear Unit (ReLU). `forward(z)` menghasilkan output dengan mengambil nilai maksimum antara 0 dan z sehingga hanya menghasilkan nilai positif, sedangkan `backward(z)` menghasilkan gradien berupa 1 jika $z > 0$ dan 0 jika $z \leq 0$.

`SigmoidActivation` mengimplementasikan fungsi sigmoid yang mengubah input menjadi rentang antara 0 dan 1. `forward(z)` menghitung $\sigma(z) = \frac{1}{1+\exp(-z)}$, sedangkan `backward(z)` menghitung turunan sigmoid dengan rumus $\sigma(z)(1 - \sigma(z))$ menggunakan nilai output dari method `forward`.

`TanhActivation` mengimplementasikan fungsi tanh (hiperbolik tangen) yang mengubah input ke rentang -1 sampai 1. `forward(z)` menghitung $\tanh(z)$ dan `backward(z)` mengembalikan turunan fungsi tanh yaitu $1 - \tanh(z)^2$.

SoftmaxActivation aktivasi softmax umumnya digunakan pada layer output untuk masalah klasifikasi multikelas. $\text{forward}(z)$ menghitung softmax secara baris (untuk setiap sample) dengan menormalkan nilai eksponensial sehingga jumlah semua nilai output menjadi 1 dan $\text{backward}(z)$ menghitung turunan softmax. Implementasi ini menghitung $s(1 - s)$.

LeakyReLUActivation merupakan variasi dari ReLU yang mengizinkan gradien kecil untuk nilai negatif agar tidak mengalami “dead neurons”.

- Atribut: alpha (nilai default 0.01) yang menentukan kemiringan pada bagian negatif
- $\text{forward}(z)$: Mengembalikan nilai maksimum antara $\alpha \times z$ dan z
- $\text{backward}(z)$: Menghasilkan gradien 1 untuk $z > 0$ dan α untuk $z < 0$

ELUActivation (Exponential Linear Unit Activation) memberikan nilai linear pada input positif dan nilai eksponensial yang diturunkan pada input negatif.

- Atribut: alpha (nilai default 1.0) yang menentukan skala pada bagian negatif.
- $\text{forward}(z)$: Mengembalikan z untuk $z > 0$ dan $\alpha \times (\exp(z) - 1)$ untuk $z \leq 0$
- $\text{backward}(z)$: Menghitung turunan ELU, yaitu 1 untuk $z > 0$ dan $\alpha \times \exp(z)$ untuk $z \leq 0$

2. initializers.py

File ini tidak memiliki kelas dan hanya menyimpan sebuah fungsi yang mengembalikan bobot (W) dan bias (b) untuk sebuah layer dengan dimensi input dan output tertentu dengan metode inisialisasi yang dipilih.

- Parameter
 1. `input_dim` (integer): Menentukan jumlah neuron/input pada layer sebelumnya yang menjadi masukan untuk layer saat ini.
 2. `output_dim` (integer): Menentukan jumlah neuron pada layer saat ini.
 3. `method` (string): Menentukan metode inisialisasi yang akan digunakan. Nilai default adalah "random_uniform". Metode yang tersedia meliputi:

- a. "zero": Inisialisasi semua bobot dan bias dengan nol
 - b. "random_uniform": Inisialisasi bobot dan bias dengan nilai acak yang diambil secara uniform dalam rentang tertentu (default -0.1 hingga 0.1)
 - c. "random_normal": Inisialisasi dengan distribusi normal berdasarkan mean dan varians yang diberikan
 - d. "xavier": Inisialisasi dengan metode Xavier yang menyesuaikan rentang nilai berdasarkan jumlah neuron input dan output
 - e. "he": Inisialisasi dengan metode He yang menyesuaikan nilai bobot secara acak dengan faktor skala berdasarkan jumlah neuron input
4. init_params (dictionary): Parameter tambahan yang digunakan oleh masing-masing metode inisialisasi. Misalnya, untuk "random_uniform" bisa mencakup nilai lower dan upper sebagai batas bawah dan atas serta seed untuk pengacakan yang dapat diatur ulang.

- Proses dan Metode

Fungsi memeriksa nilai parameter method untuk memilih cara inisialisasi:

1. "zero": Mengembalikan array bobot dan bias yang seluruh elemennya nol.
2. "random_uniform": Menggunakan fungsi `np.random.uniform` untuk menghasilkan bobot dan bias secara acak dalam rentang yang telah ditentukan.
3. "random_normal": Menggunakan `np.random.normal` untuk menghasilkan bobot dan bias dengan distribusi normal berdasarkan parameter mean dan var.
4. "xavier": Menggunakan rumus $\sqrt{\frac{6}{(input_{dim} + output_{dim})}}$ untuk menentukan batas rentang nilai, kemudian menggunakan distribusi uniform di antara nilai negatif dan positif batas tersebut. Bias diinisialisasi dengan nol.

5. "he": Menggunakan rumus $\sqrt{\frac{2}{input_dim}}$ untuk mengalikan output dari distribusi normal, bias diinisialisasi dengan nol. Jika method yang diberikan tidak dikenali, fungsi akan melempar error.

- Keluaran

Fungsi ini mengembalikan sebuah tuple (W, b), di mana W adalah array numpy berukuran (input_dim, output_dim) yang menyimpan bobot dan b merupakan array numpy berukuran (1, output_dim) yang menyimpan bias.

3. losses.py

Kelas Loss merupakan kelas abstrak untuk mendefinisikan semua fungsi loss dalam model. Kelas ini tidak memiliki atribut, tetapi terdapat dua method abstrak yaitu `loss(y_true, y_pred)` untuk menghitung nilai loss serta `gradient(y_true, y_pred)` untuk menghitung gradien dari fungsi loss. Keduanya harus diimplementasikan oleh kelas turunannya.

Kelas `MSELoss` merupakan implementasi pertama dari kelas Loss untuk fungsi Mean Squared Error (MSE). Pada method `loss`, `MSELoss` menghitung rata-rata kuadrat selisih antara nilai aktual (`y_true`) dan prediksi (`y_pred`). Method `gradient` menghitung turunan MSE dengan rumus $2 \times \frac{(y_{pred} - y_{true})}{jumlah\ y_{true}}$ untuk mengarahkan update bobot pada saat training.

Kelas `BinaryCrossEntropy` mengimplementasikan fungsi loss untuk masalah klasifikasi biner. Pada method `loss`, nilai prediksi di-clip menggunakan `np.clip` untuk menghindari perhitungan $\log(0)$ yang menghasilkan nilai tak terhingga, lalu menghitung loss sebagai nilai negatif rata-rata dari kombinasi $y_{true} \times \log(y_{pred})$ dan $(1 - y_{true}) \times \log(1 - y_{pred})$. Method `gradient` menghitung gradien loss berdasarkan rumus diferensial dari fungsi cross entropy biner dengan normalisasi berdasarkan jumlah sampel.

Kelas `CategoricalCrossEntropy` digunakan untuk fungsi loss pada masalah klasifikasi multikelas. Pada method `loss`, nilai prediksi juga di-clip untuk menghindari masalah logaritma dari 0, kemudian loss dihitung sebagai rata-rata negatif dari jumlah (sum) perkalian antara nilai `y_true` dan log dari

nilai prediksi y_{pred} untuk setiap sampel, sedangkan method gradient mengembalikan selisih antara y_{pred} dan y_{true} yang dibagi dengan jumlah sampel yang merupakan turunan loss untuk update parameter model.

4. model.py

Pada file ini, terdapat dua buah kelas yaitu kelas layer dan FFNN. Kelas Layer merepresentasikan satu lapisan dalam jaringan neuron. Saat inisialisasi, kelas ini menerima parameter seperti input_dim dan output_dim (dimensi input dan output), objek dari fungsi aktivasi yang akan digunakan, metode inisialisasi bobot, parameter inisialisasi tambahan, opsi penggunaan RMS Normalization (use_rms_norm), serta parameter regularisasi (l1_lambda dan l2_lambda). Atribut dalam kelas ini antara lain:

- W dan b: Bobot dan bias yang diinisialisasi menggunakan fungsi initialize_weights() sesuai dengan metode yang dipilih.
- dW dan db: Gradien dari bobot dan bias yang dihitung selama proses backpropagation.
- activation: Objek fungsi aktivasi (misalnya ReLU, Sigmoid, Tanh) yang menentukan transformasi non-linear dari input.
- X, Z, A: Variabel penyimpanan nilai input (X), nilai linear sebelum aktivasi (Z), dan output setelah aktivasi (A) pada saat forward propagation.
- rms_norm: Objek RMSNorm (jika digunakan) untuk melakukan normalisasi pada output layer.

Method utama yang ada adalah:

- forward(X): Menghitung nilai linear ($Z = X \cdot W + b$) kemudian mengaplikasikan fungsi aktivasi untuk menghasilkan output (A). Jika RMS Normalization diaktifkan, output akan dinormalisasi lebih lanjut.
- backward(dA): Menghitung gradien dari error (dA) yang diterima, mengaplikasikan turunan fungsi aktivasi, dan menghitung gradien terhadap parameter W dan b dengan termasuk penambahan regularisasi L1 dan L2 jika diperlukan.
- update(learning_rate): Memperbarui nilai W dan b dengan menggunakan gradien yang telah dihitung, dikalikan dengan laju pembelajaran, serta memperbarui parameter RMSNorm bila aktif.

Kelas FFNN (Feedforward Neural Network) menggabungkan beberapa objek Layer untuk membentuk jaringan neuron feedforward. Kelas ini mengelola seluruh arsitektur model, proses forward propagation, backpropagation, dan pembaruan parameter jaringan secara iteratif. Atribut utama meliputi:

- `layers`: Daftar objek Layer yang menyusun jaringan, diinisialisasi berdasarkan konfigurasi layer yang diterima.
- `loss_func`: Fungsi loss (seperti `MSELoss`, `BinaryCrossEntropy`, atau `CategoricalCrossEntropy`) yang digunakan untuk menghitung error antara output jaringan dan nilai target.
- `l1_lambda` dan `l2_lambda`: Nilai regularisasi untuk L1 dan L2 yang diterapkan pada setiap layer selama backpropagation.

Method penting pada kelas FFNN antara lain:

- `init(layers_config, loss, weight_init_method, init_params, l1_lambda, l2_lambda)`: Mengonfigurasi dan membuat setiap layer berdasarkan parameter yang diterima seperti pemilihan fungsi aktivasi dan opsi normalisasi.
- `forward(X)`: Mengoperasikan input melalui setiap layer secara berurutan untuk menghasilkan output akhir jaringan.
- `backward(y_true, y_pred)`: Menghitung gradien error menggunakan fungsi loss dan menerapkan backpropagation dari layer terakhir ke layer pertama untuk mengupdate gradien masing-masing layer.
- `update(learning_rate)`: Mengiterasi setiap layer dan memanggil method `update()` untuk memperbarui bobot dan bias berdasarkan gradien yang telah dihitung.
- `_calculate_reg_loss()`: Menghitung total loss dari regularisasi L1 dan L2 yang diterapkan pada seluruh bobot jaringan.
- `train(X_train, y_train, X_val=None, y_val, epochs, batch_size, learning_rate, verbose)`: Melatih model menggunakan data training, melakukan proses forward, menghitung loss, backpropagation, dan pembaruan parameter secara iteratif selama sejumlah epoch. Fungsi ini juga menyediakan visualisasi loss training (dan validasi) menggunakan `matplotlib`.

- `summary()`: Menampilkan ringkasan struktur jaringan seperti dimensi input dan output serta tipe fungsi aktivasi di setiap layer.
- `visualize()`, `plot_gradients_dist()`, `plot_weights_dist()`: Method untuk memvisualisasikan struktur jaringan, distribusi gradien, dan distribusi bobot.

5. `norm.py`

Kelas `RMSNorm` bertujuan untuk melakukan normalisasi berdasarkan Root Mean Square (RMS) pada data input.

Atribut:

- `dim`: Menyimpan nilai integer dari dimensi fitur dari input yang akan dinormalisasi
- `eps`: Nilai epsilon (default $1e-8$) yang digunakan untuk mencegah pembagian dengan nol ketika menghitung akar kuadrat dari rata-rata kuadrat nilai input
- `scale`: Parameter skala yang disimpan dalam bentuk array numpy dengan ukuran $(1, \text{dim})$. Nilai awalnya adalah ones, dan parameter ini berperan untuk mengalikan hasil normalisasi
- `dscale`: Array numpy dengan ukuran yang sama dengan `scale` untuk menyimpan gradien terhadap parameter `scale` selama proses backpropagation

Method:

- `init(self, dim, eps=1e-8)`: Konstruktor yang menginisialisasi objek `RMSNorm` dengan nilai dimensi (`dim`), epsilon (`eps`), serta mengatur nilai awal `scale` (diisi dengan ones) dan `dscale` (diisi dengan nol)
- `forward(self, x)`: Method ini menerima input `x` dan melakukan proses normalisasi
 - Simpan `x` dalam atribut `self.x` untuk digunakan pada tahap backward
 - Hitung nilai RMS untuk setiap sampel dengan cara mengambil akar dari rata-rata kuadrat nilai-nilai dalam `x` ditambah dengan `eps`
 - Bagi `x` dengan nilai RMS tersebut untuk mendapatkan data yang ternormalisasi
 - Kalikan hasil normalisasi dengan parameter `scale` dan mengembalikan hasilnya

- `backward(self, grad_output)`: Method ini berfungsi untuk menghitung gradien dari fungsi loss sehubungan dengan input `x` melalui operasi normalisasi
 - Hitung gradien terhadap parameter `scale` dan menyimpannya di `self.dscale`
 - Kemudian, hitung gradien `dx` langsung dengan membagi gradien keluaran dengan RMS dan dikalikan dengan `scale`
 - Hitung gradien terhadap nilai RMS (`drms`) dari input `x` dan mengkonversinya menjadi gradien tambahan `dx_rms`
 - Kembalikan total gradien sebagai penjumlahan `dx` dan `dx_rms`
- `update(self, learning_rate)`: Method ini digunakan untuk memperbarui parameter `scale` berdasarkan gradien yang telah dihitung (`dscale`) dan laju pembelajaran (`learning_rate`)
 - Parameter `scale` diperbarui dengan mengurangi hasil perkalian laju pembelajaran dan gradien `dscale`.

6. `utils.py`

File ini tidak memiliki kelas dan hanya menyimpan 5 fungsi untuk memvisualisasi jaringan, plotting distribusi bobot, plotting distribusi gradien, menyimpan serta memuat model.

b. Penjelasan Forward Propagation

Pada kelas `FFNN` metode `forward(X)` menerima input `X` dan kemudian mengoperasikan data tersebut ke setiap layer yang terdapat dalam daftar `self.layers`. Pada tiap layer, proses forward propagation terjadi sebagai berikut:

1. Perhitungan Linear (Pre-Activation)

Input `X` pada layer tertentu dikalikan dengan matriks bobot `W` dan ditambahkan dengan bias `b` untuk menghasilkan nilai linear `Z`, yaitu:

$$Z = X \times W + b$$

Nilai `Z` ini merupakan representasi linear dari input yang masuk ke layer tersebut.

2. Fungsi Aktivasi (Non-Linear Transformation)

Nilai `Z` kemudian diproses oleh fungsi aktivasi yang telah didefinisikan pada objek layer melalui `self.activation.forward(Z)`. Fungsi aktivasi (misalnya ReLU, Sigmoid, Tanh, atau Softmax) mengaplikasikan transformasi non-linear

pada Z untuk menghasilkan output aktivasi A . Proses non-linear ini berguna untuk mempelajari hubungan antara input dan output yang tidak dapat dipisahkan secara linear.

3. RMS Normalization (Opsional)

Jika opsi RMS Normalization diaktifkan (`use_rms_norm == True`), maka output aktivasi A akan diproses lebih lanjut oleh modul `RMSNorm`. `RMSNorm` menghitung nilai akar rata-rata kuadrat dari A dan menormalkan nilai-nilai tersebut, lalu mengalikannya dengan parameter skala yang dapat di-update selama pelatihan.

4. Propagasi ke Layer Berikutnya

Output A dari layer pertama menjadi input untuk layer berikutnya. Proses perhitungan linear dan aktivasi diulangi pada setiap layer secara berurutan. Dengan demikian, informasi dari data input diproses secara bertingkat melalui kombinasi transformasi linear dan nonlinear.

5. Output Akhir

Setelah data melewati seluruh layer, output dari layer terakhir merupakan hasil akhir dari forward propagation yang nantinya digunakan untuk membuat prediksi. Output ini dapat diteruskan ke fungsi loss untuk perhitungan error dan proses backpropagation selanjutnya.

c. Penjelasan Backward Propagation dan Weight Update

Proses ini dimulai dengan perhitungan error atau loss antara output jaringan (hasil forward propagation) dengan nilai target yang sebenarnya, lalu dikonversi menjadi gradien melalui fungsi loss. Jika lapisan terakhir menggunakan aktivasi Softmax dengan fungsi loss Categorical Cross-Entropy, gradien output dapat disederhanakan menjadi $y_{pred} - y_{true}$ sebagai representasi error karena hubungan matematis khusus antara Softmax dan cross-entropy yang menghasilkan gradien tersebut.

Setelah gradien awal diperoleh, backward propagation berjalan dengan cara mengalirkan gradien tersebut dari layer terakhir ke layer pertama menggunakan aturan rantai (chain rule). Pada tiap layer, method backward dari kelas `Layer` mengambil gradien dari layer berikutnya (dA) dan mengalikannya dengan turunan fungsi aktivasi pada nilai pre-aktivasi (Z) untuk menghitung gradien lokal dZ . Nilai dZ ini kemudian

digunakan untuk menghitung gradien bobot dW melalui perkalian matriks antara transpose dari input X (yang disimpan selama proses forward) dengan dZ dan untuk menghitung gradien bias db dengan menjumlahkan semua elemen dZ . Pembagian dengan jumlah sampel memastikan bahwa gradien dinormalisasi sehingga pembaruan parameter menjadi stabil. Selain itu, jika regularisasi L1 atau L2 diterapkan, gradien dW juga diperbaiki dengan menambahkan kontribusi dari masing-masing regularisasi tersebut (penjumlahan $\lambda_1 \cdot \text{sign}(W)$ untuk L1 dan $\lambda_2 \cdot W$ untuk L2). Setelah perhitungan gradien lokal selesai, setiap layer mengembalikan gradien input dX yang diperoleh dari perkalian dZ dengan transpose bobot W , gradien ini kemudian diteruskan ke layer sebelumnya sehingga seluruh jaringan secara berurutan mendapatkan update gradien.

Setelah seluruh gradien dihitung melalui backward propagation, tahap weight update dilakukan. Pada tahap ini, method update dari setiap layer dipanggil, yang bertugas memperbarui parameter bobot W dan bias b dengan mengurangi hasil perkalian antara gradien yang telah dihitung dan laju pembelajaran (learning rate). Secara matematis, pembaruan dilakukan dengan rumus:

$$W \leftarrow W - \text{learning_rate} \times dW$$

$$b \leftarrow b - \text{learning_rate} \times db$$

Jika opsi RMS Normalization diaktifkan, parameter skala pada RMSNorm juga diperbarui sesuai dengan gradien yang relevan. Pembaruan ini dilakukan secara iteratif setiap kali batch data diproses selama training untuk meminimalkan fungsi loss.

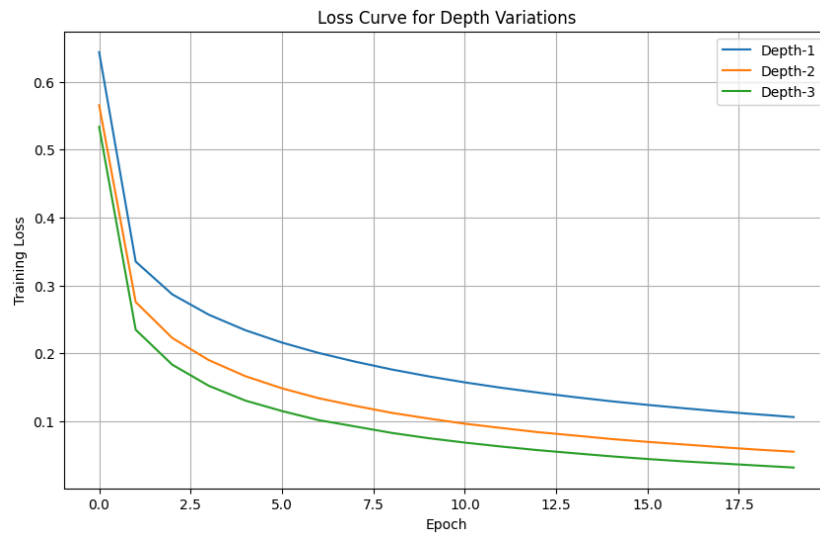
B. Hasil Pengujian

a. Pengaruh Depth dan Width

Berdasarkan hasil pengujian terhadap variasi depth, depth yang lebih besar cenderung menghasilkan akurasi model yang lebih baik daripada depth yang lebih kecil, terbukti dari depth-1 yang memberikan akurasi sebesar 96,3071%, depth-2 sebesar 97,2357%, dan depth-3 sebesar 97,3429%. Loss yang dihasilkan juga cenderung menurun ketika depth yang diberikan semakin besar dan epoch yang dijalankan semakin banyak.

Tabel 2.2.1.1 Akurasi Variasi Depth

Depth	Akurasi
1	96,3071%
2	97,2357%
3	97,3429%

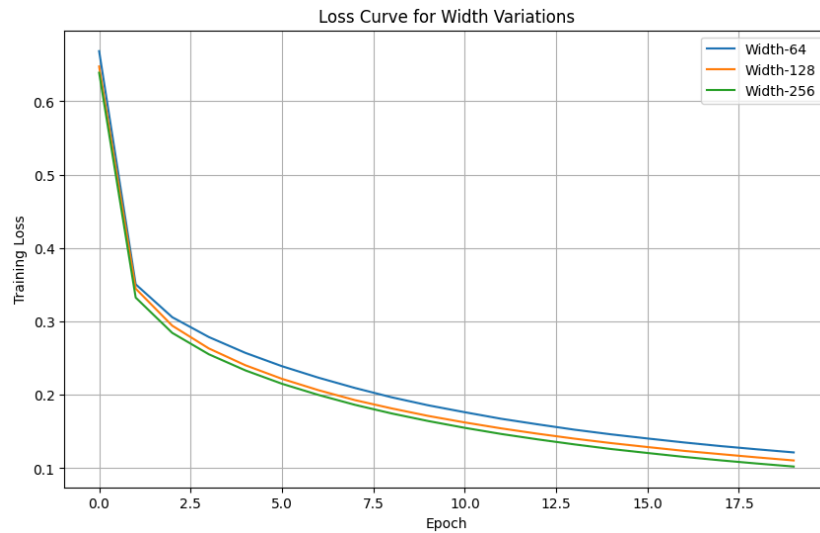


Gambar 2.2.1.1 Kurva Loss Variasi Depth

Hal ini juga berlaku untuk variasi width, semakin besar width maka semakin besar juga akurasi yang dihasilkan. Pengujian dengan width sebesar 64 memberikan akurasi sebesar 95,9357%, width-128 sebesar 96,3071%, dan width-256 sebesar 96,4571%. Loss yang dihasilkan dari variasi width juga mengikuti pola milik variasi depth. Akan tetapi, penambahan ukuran depth atau width ini akan berdampak kepada penambahan kebutuhan komputasi yang juga akan mempengaruhi lama pemrosesan model.

Tabel 2.2.1.2 Akurasi Variasi Width

Width	Akurasi
64	95,9357%
128	96,3071%
256	96,4571%



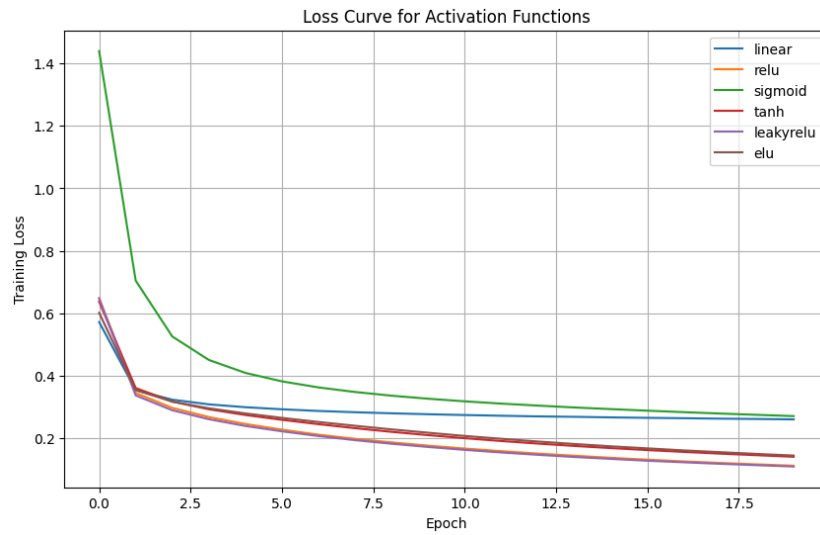
Gambar 2.2.1.2 Kurva Loss Variasi Width

b. Pengaruh Fungsi Aktivasi

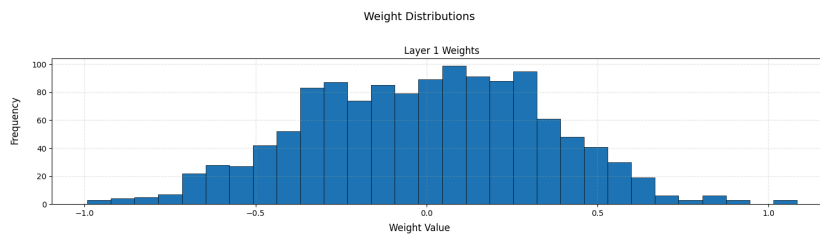
Dari keenam jenis fungsi aktivasi yang diuji seperti Linear, ReLU, Sigmoid, Tanh, LeakyReLU, dan ELU. LeakyReLU memiliki akurasi terbaik sebesar 96,3929%, lalu ReLU sebesar 96,3143%, ELU sebesar 95,4%, Tanh sebesar 95,3929%, Linear sebesar 92,0643%, dan Sigmoid sebesar 91,9571%. Pola terhadap training loss juga memiliki urutan yang sama seperti akurasi dengan LeakyReLU yang memiliki loss error terkecil dan Sigmoid dengan loss error terbesar.

Tabel 2.2.2.1 Akurasi Variasi Fungsi Aktivasi

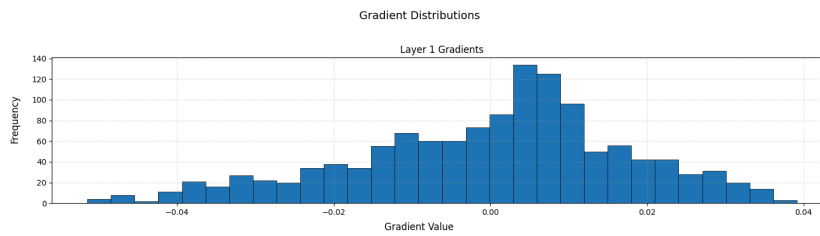
Fungsi Aktivasi	Akurasi
Sigmoid	91,9571%
Linear	92,0643%
Tanh	95,3929%
ELU	95,4%
ReLU	96,3143%
LeakyReLU	96,3929%



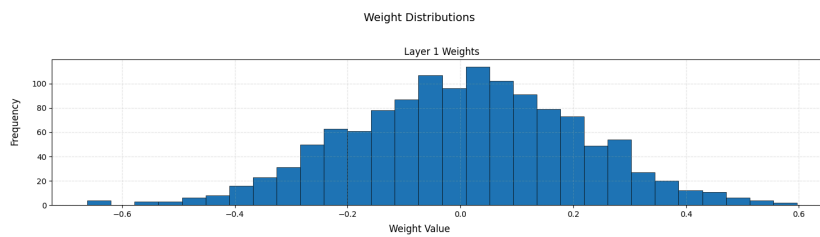
Gambar 2.2.2.1 Kurva Loss Variasi Fungsi Aktivasi



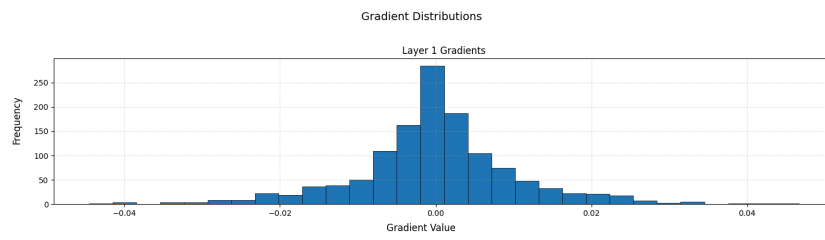
Gambar 2.2.2.2 Distribusi Bobot Sigmoid pada Layer-1



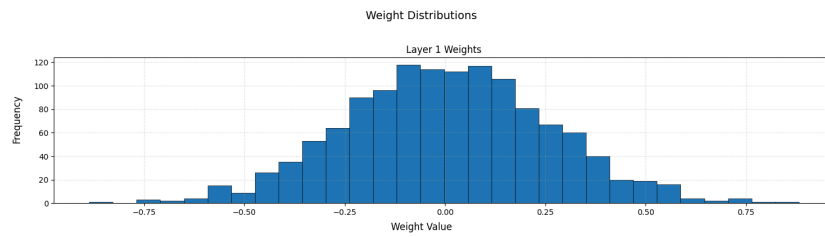
Gambar 2.2.2.3 Distribusi Gradien Sigmoid pada Layer-1



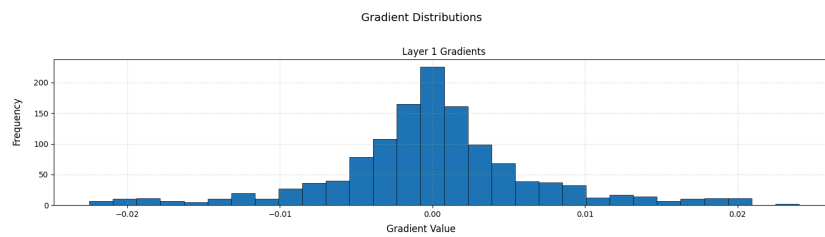
Gambar 2.2.2.4 Distribusi Bobot Linear pada Layer-1



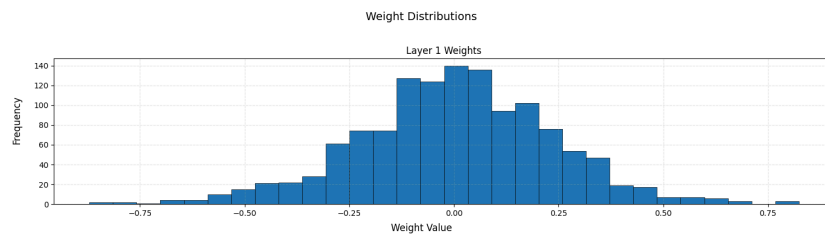
Gambar 2.2.2.5 Distribusi Gradien Linear pada Layer-1



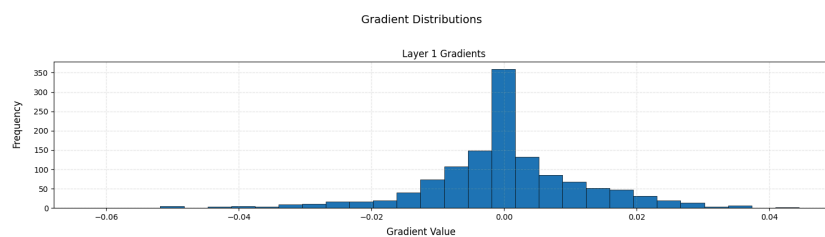
Gambar 2.2.2.6 Distribusi Bobot Tanh pada Layer-1



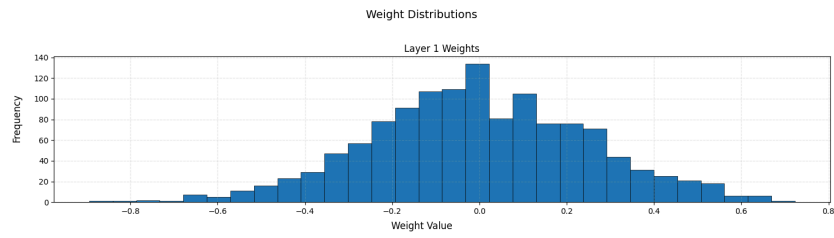
Gambar 2.2.2.7 Distribusi Gradien Tanh pada Layer-1



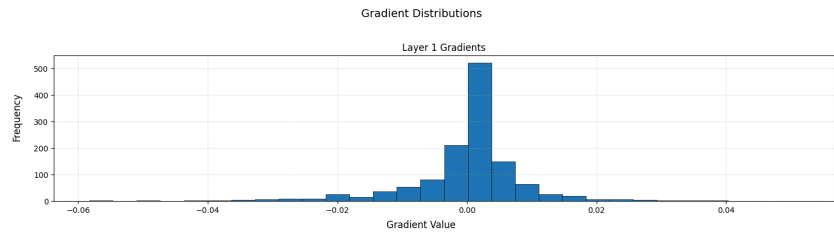
Gambar 2.2.2.8 Distribusi Bobot ELU pada Layer-1



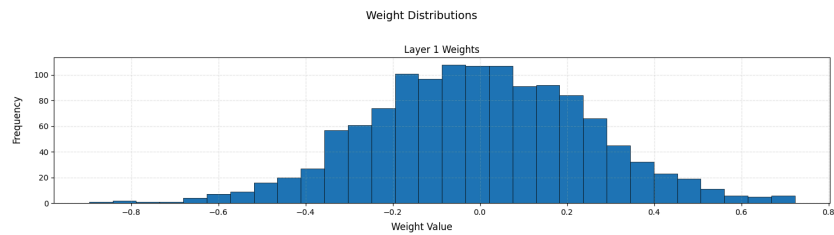
Gambar 2.2.2.9 Distribusi Gradien ELU pada Layer-1



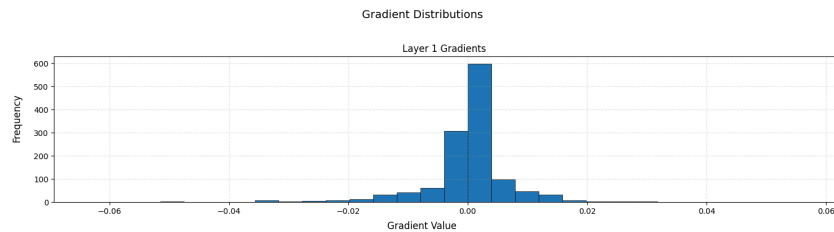
Gambar 2.2.2.10 Distribusi Bobot ReLU pada Layer-1



Gambar 2.2.2.11 Distribusi Gradien ReLU pada Layer-1



Gambar 2.2.2.12 Distribusi Bobot LeakyReLU pada Layer-1



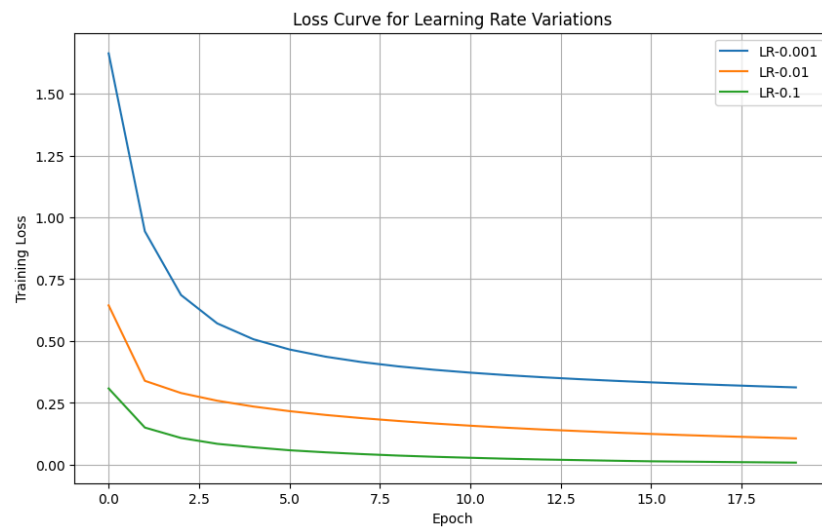
Gambar 2.2.2.13 Distribusi Gradien LeakyReLU pada Layer-1

c. Pengaruh Learning Rate

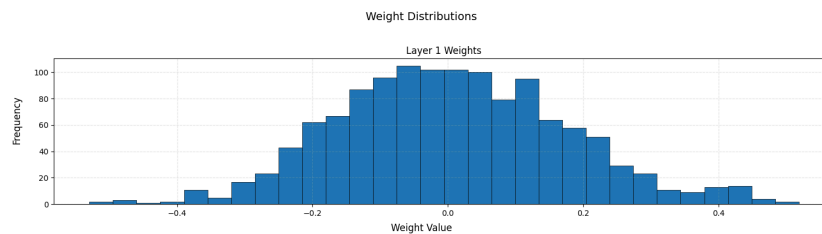
Hasil dari pengujian dengan learning rate sebesar 0,001, 0,01, dan 0,1 memberikan akurasi yang semakin besar secara berturut-turut dan training loss yang semakin kecil. Learning rate yang besar dapat mempercepat proses pelatihan, namun hal ini juga dapat menyebabkan ketidakstabilan karena model melewati titik minimum optimal atau divergensi pada kasus terburuk.

Tabel 2.2.3.1 Akurasi Variasi Learning Rate

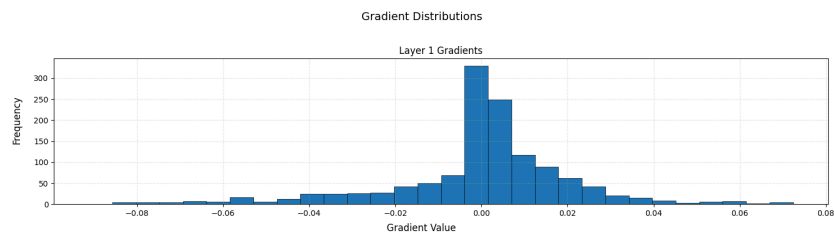
Learning Rate	Akurasi
0,001	91,0571%
0,01	96,3786%
0,1	97,9286%



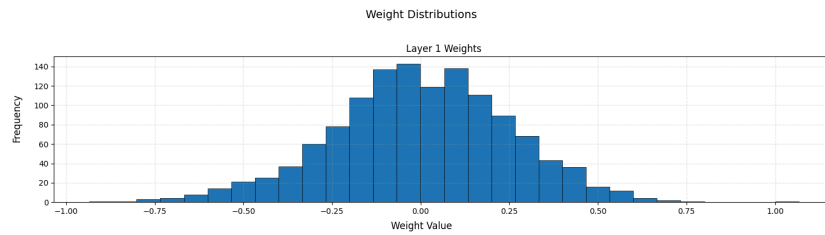
Gambar 2.2.3.1 Kurva Loss Variasi Learning Rate



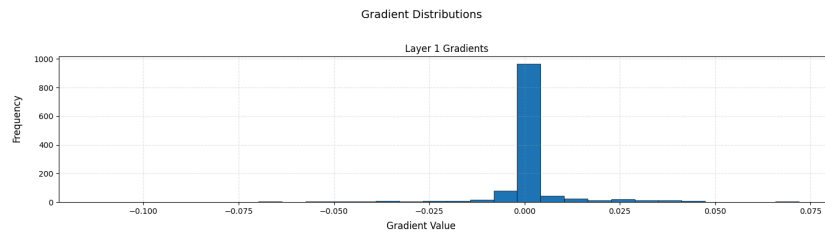
Gambar 2.2.3.2 Distribusi Bobot Learning rate 0,001



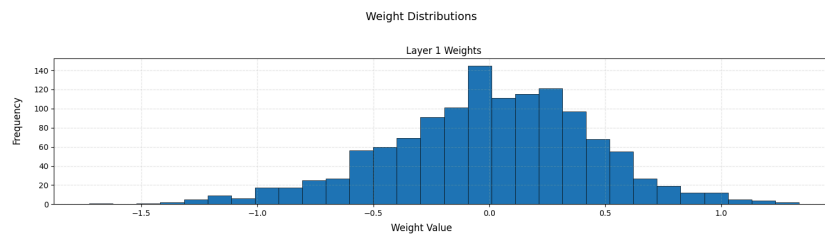
Gambar 2.2.3.3 Distribusi Gradien Learning rate 0,001



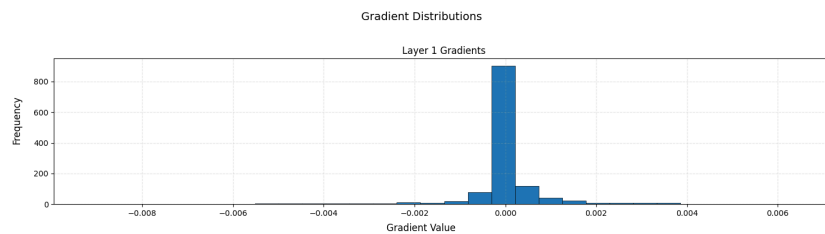
Gambar 2.2.3.4 Distribusi Bobot Learning rate 0,01



Gambar 2.2.3.5 Distribusi Gradien Learning rate 0,01



Gambar 2.2.3.6 Distribusi Bobot Learning rate 0,1



Gambar 2.2.3.7 Distribusi Gradien Learning rate 0,1

d. Pengaruh Inisialisasi Bobot

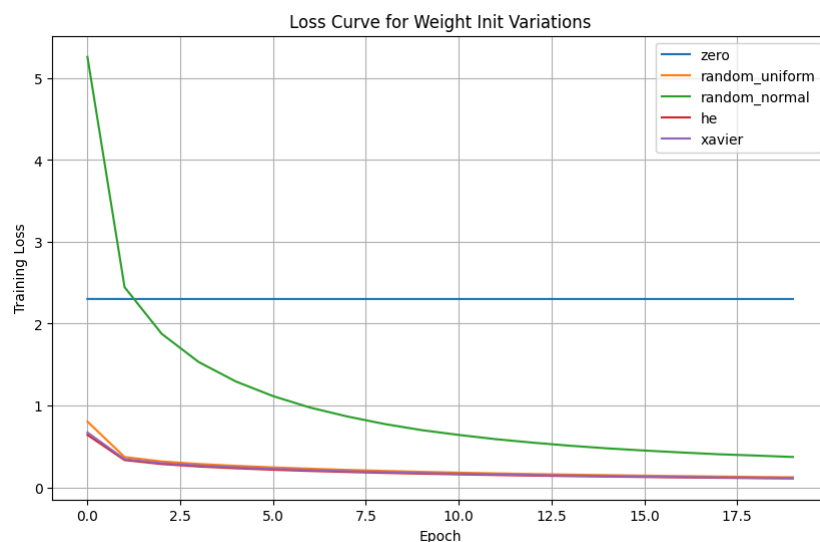
Pengujian variasi inisialisasi bobot menggunakan metode zero, random_uniform, random_normal, he, dan xavier. Dari kelima variasi tersebut, akurasi terurut dari kecil ke besar adalah zero, random_normal, random_uniform, xavier, dan he dengan nilai akurasi masing-masing sebesar 11,3857%, 90,3571%, 95,9857%, 96,2071%, dan 96,2857%.

Pada gambar kurva loss di bawah, terlihat bahwa inisialisasi bobot dengan zero memiliki nilai loss yang tidak berubah sama sekali dari awal hingga akhir epoch, hal ini menunjukkan bahwa model tidak mampu meningkatkan performanya karena

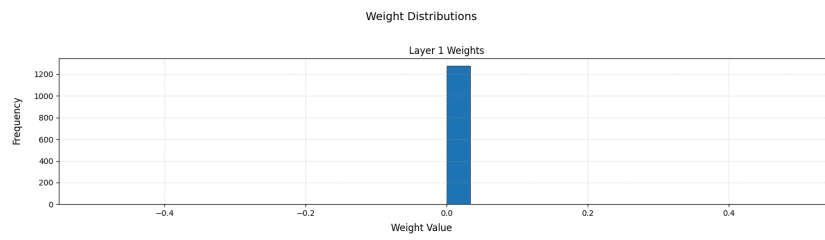
gradiennya selalu bernilai nol. Sementara itu, metode inisialisasi random_uniform, random_normal, He, dan Xavier sama-sama menunjukkan penurunan training loss yang menandakan proses pembelajaran berjalan secara positif. Pada awal pelatihan, inisialisasi random_normal cenderung memiliki loss yang lebih tinggi namun kemudian turun cukup tajam, sedangkan random_uniform, He, dan Xavier memulai dari nilai loss yang relatif lebih rendah. Secara keseluruhan, metode He dan Xavier lebih stabil dan cepat dalam mencapai loss yang rendah dibandingkan metode inisialisasi lainnya.

Tabel 2.2.4.1 Akurasi Variasi Insialisasi Bobot

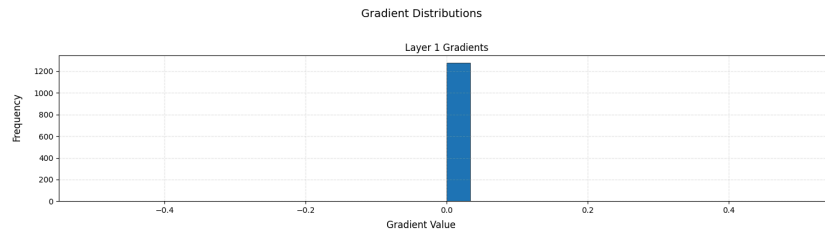
Inisialisasi Bobot	Akurasi
zero	11,3857%
random_normal	90,3571%
random_uniform	95,9857%
xavier	96,2071%
he	96,2857%



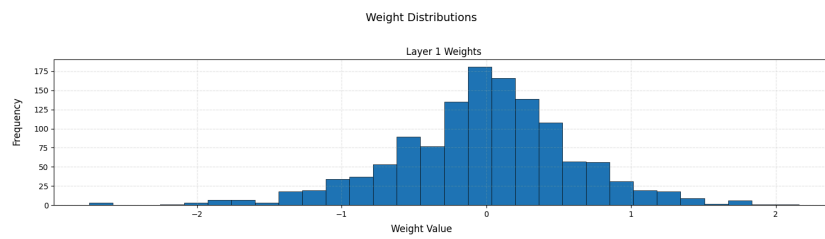
Gambar 2.2.4.1 Kurva Loss Variasi Inisialisasi Bobot



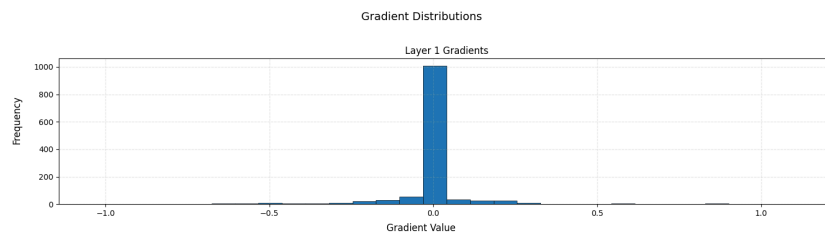
Gambar 2.2.4.2 Distribusi Bobot Inisialisasi Zero pada Layer-1



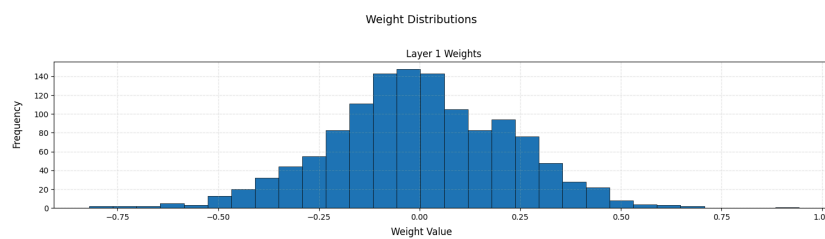
Gambar 2.2.4.3 Distribusi Gradien Inisialisasi Zero pada Layer-1



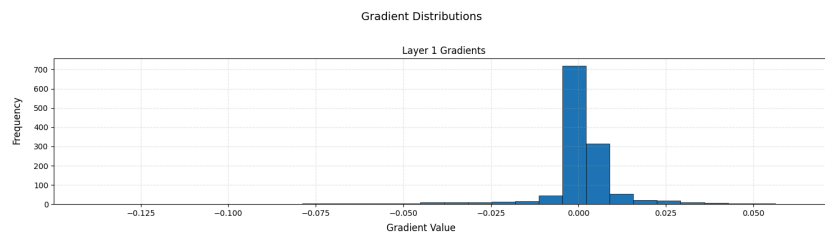
Gambar 2.2.4.4 Distribusi Bobot Inisialisasi Random_normal pada Layer-1



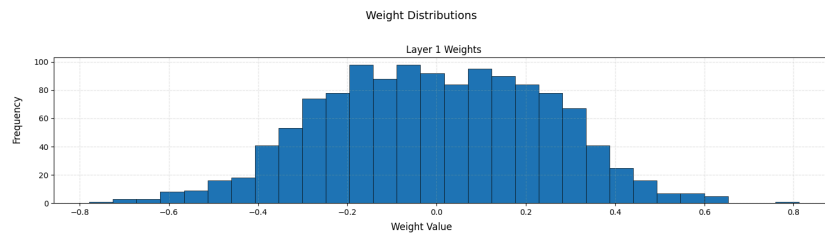
Gambar 2.2.4.5 Distribusi Gradien Inisialisasi Random_normal pada Layer-1



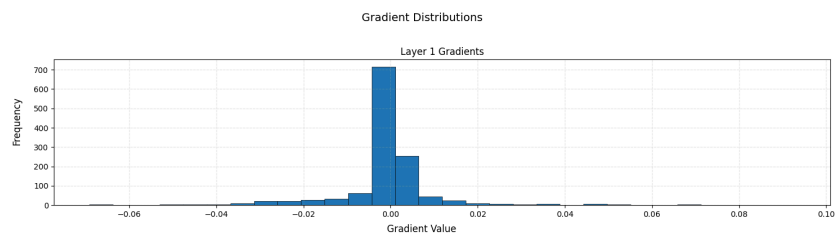
Gambar 2.2.4.6 Distribusi Bobot Inisialisasi Random_uniform pada Layer-1



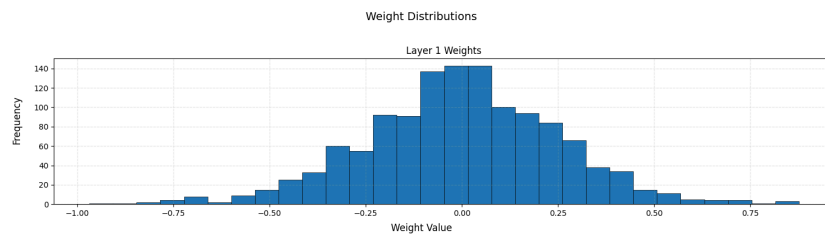
Gambar 2.2.4.7 Distribusi Gradien Inisialisasi Random_uniform pada Layer-1



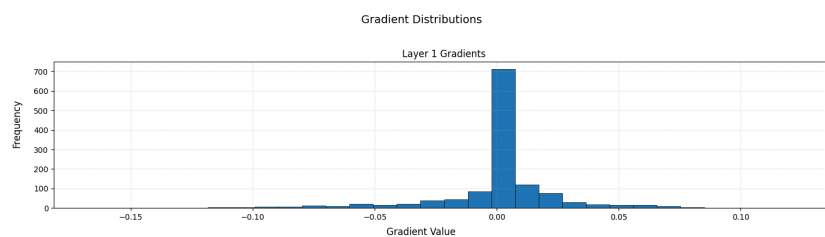
Gambar 2.2.4.8 Distribusi Bobot Inisialisasi Xavier pada Layer-1



Gambar 2.2.4.9 Distribusi Gradien Inisialisasi Xavier pada Layer-1



Gambar 2.2.4.10 Distribusi Bobot Inisialisasi He pada Layer-1



Gambar 2.2.4.11 Distribusi Gradien Inisialisasi He pada Layer-1

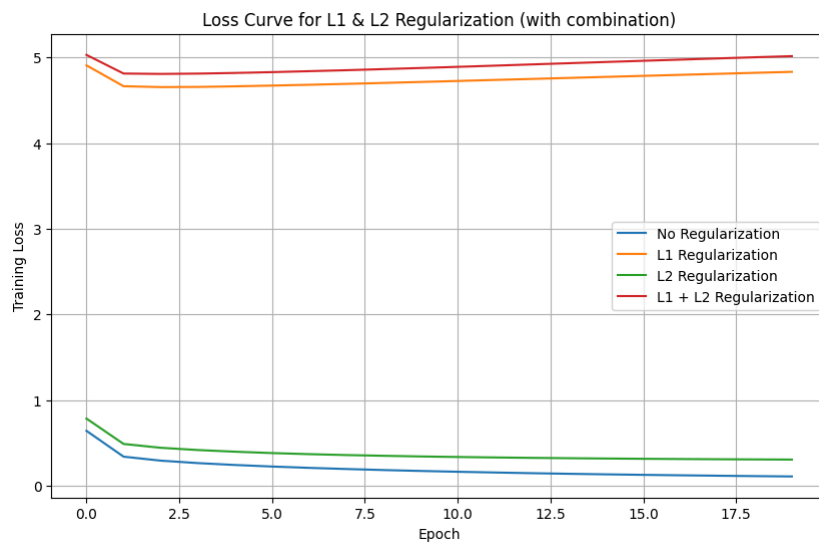
e. Pengaruh Regularisasi

Dengan membandingkan antara tidak menggunakan regularisasi sama sekali, hanya menggunakan tipe L1, hanya menggunakan tipe L2, dan menggunakan kedua regularisasi. Akurasi yang dihasilkan hanya berbeda sedikit dengan nilai akurasi sebesar 96,1929%, 96,2214%, 96,3429%, dan 96,0857% secara berurut.

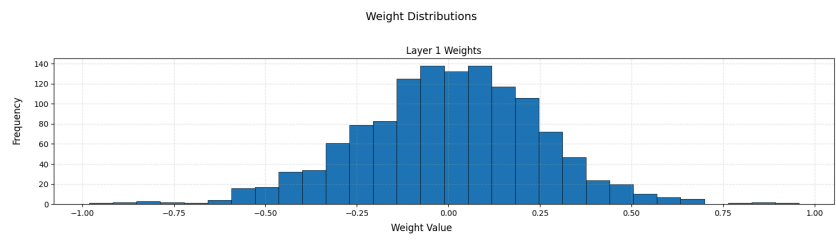
Berbeda dengan training loss yang dihasilkan, tipe L1 dan kombinasi memiliki loss yang lebih besar daripada tanpa regularisasi dan tipe L2. Pola loss yang dihasilkan oleh regularisasi L1 dan kombinasi cenderung meningkat jika epoch semakin besar, sedangkan tanpa regularisasi dan regularisasi L2 cenderung menurun. Jika diurutkan, loss terkecil dimiliki oleh tanpa regularisasi, lalu L2, L1, dan kombinasi L1 + L2.

Tabel 2.2.5.1 Akurasi Variasi Regularisasi

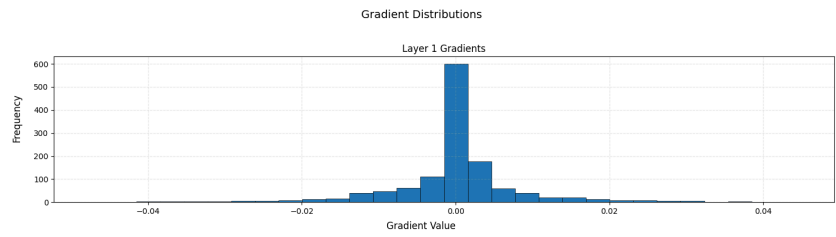
Tipe Regularisasi	Akurasi
Regularisasi L1 + L2	96,0857%
Tanpa regularisasi	96,1929%
Regularisasi L1	96,2214%
Regularisasi L2	96,3429%



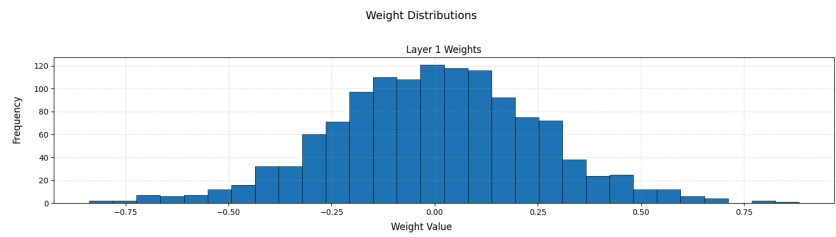
Gambar 2.2.5.1 Kurva Loss Variasi Regularisasi



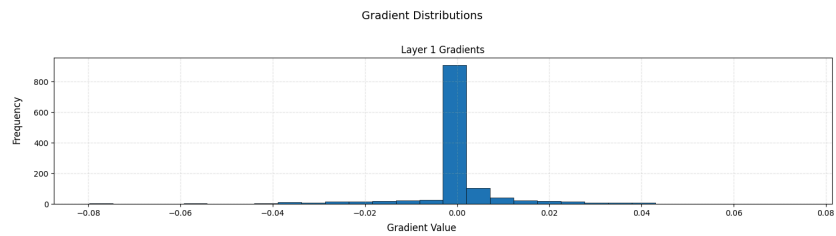
Gambar 2.2.5.2 Distribusi Bobot Regularisasi L1 + L2



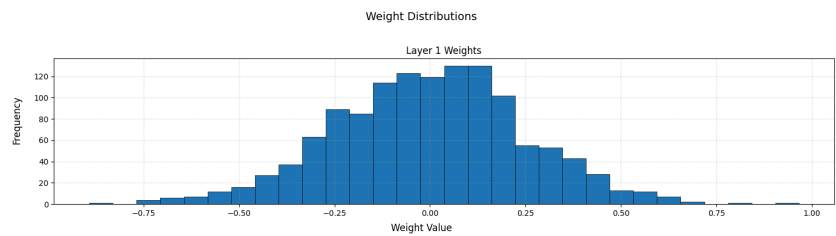
Gambar 2.2.5.3 Distribusi Gradien Regularisasi L1 + L2



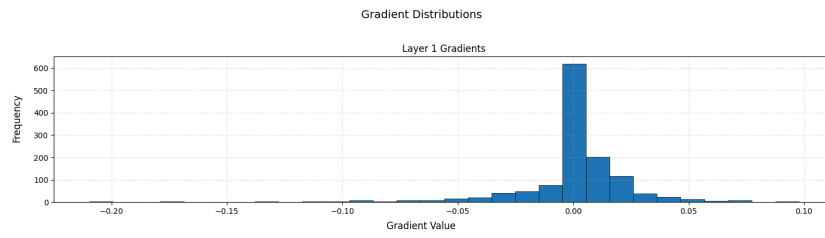
Gambar 2.2.5.4 Distribusi Bobot Tanpa Regularisasi



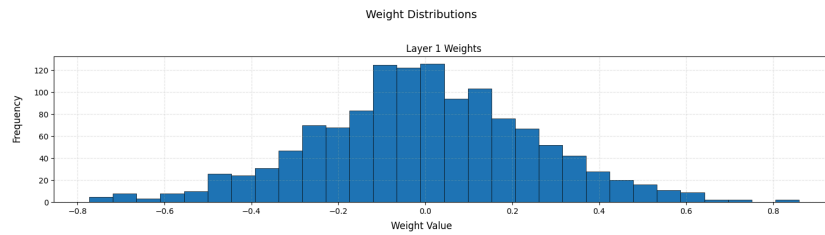
Gambar 2.2.5.5 Distribusi Gradien Tanpa Regularisasi



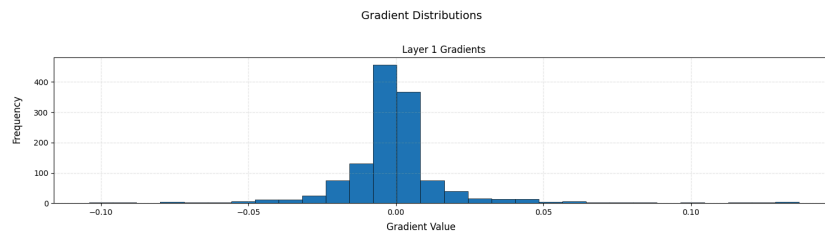
Gambar 2.2.5.6 Distribusi Bobot Regulerisasi L1



Gambar 2.2.5.7 Distribusi Gradien Regulerisasi L1



Gambar 2.2.5.8 Distribusi Bobot Regulerisasi L2



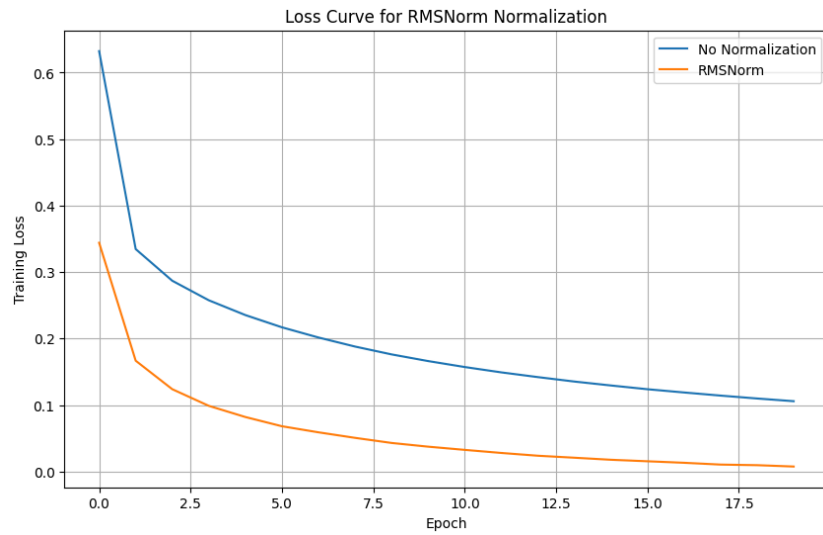
Gambar 2.2.5.9 Distribusi Gradien Regulerisasi L2

f. Pengaruh Normalisasi RMSNorm

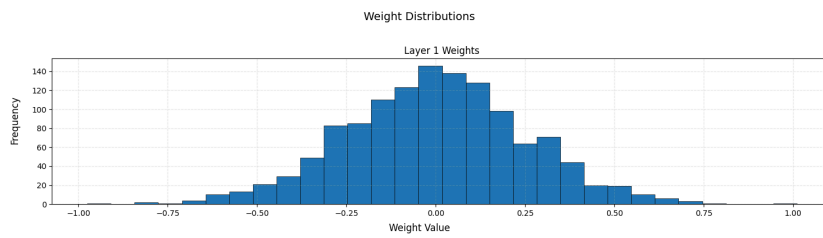
Dalam hal penggunaan RMSNorm, RMSNorm memiliki akurasi yang sedikit lebih baik sebesar 97,23% dan tanpa RMSNorm sebesar 96,34%. Loss curve yang dihasilkan juga lebih bagus dengan menggunakan RMSNorm dibandingkan tidak dari epoch 0 hingga 20.

Tabel 2.2.6.1 Akurasi Variasi Normalisasi RMSNorm

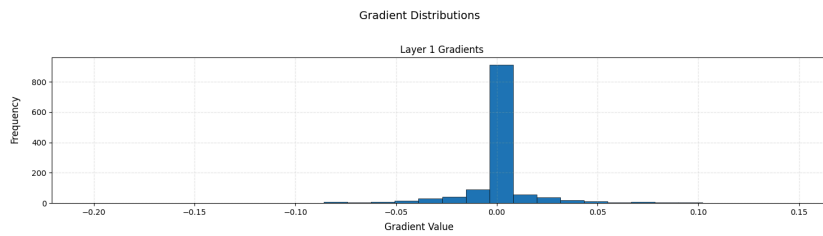
Normalisasi RMSNorm	Akurasi
Tanpa RMSNorm	96,34%
Dengan RMSNorm	97,23%



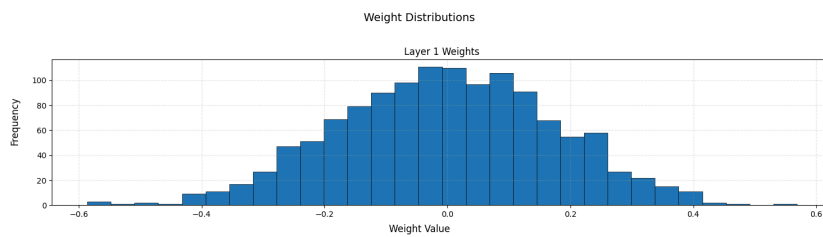
Gambar 2.2.6.1 Kurva Loss Variasi Normalisasi dengan RMSNorm



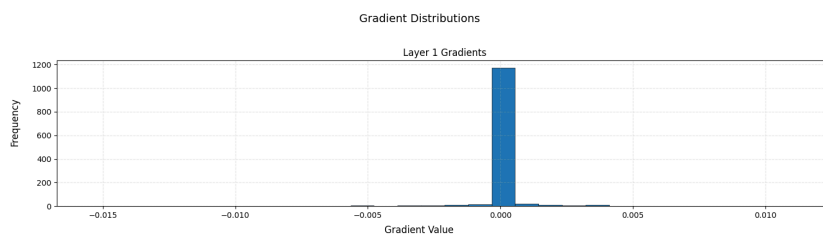
Gambar 2.2.6.2 Distribusi Bobot Tanpa Normalisasi



Gambar 2.2.6.3 Distribusi Gradien Tanpa Normalisasi




Gambar 2.2.6.4 Distribusi Bobot dengan Normalisasi RMSNorm



Gambar 2.2.6.5 Distribusi Gradien dengan Normalisasi RMSNorm

g. Perbandingan dengan Library Sklearn

Jika model ini dibandingkan dengan MLP milik sklearn, model buatan ini memiliki akurasi sebesar 96,39% yang berbeda sedikit dengan MLP sebesar 95,91%. Waktu yang digunakan antara kedua model juga sedikit berbeda dengan model buatan memakan waktu 29,91 detik, sedangkan MLP memakan waktu 44,02 detik, perbedaan yang hampir sebesar 1,5 kali lipat.

 Hasil Perbandingan Akhir
Custom FFNN Accuracy: 0.9639 (Waktu: 29.91s)
Sklearn MLP Accuracy: 0.9591 (Waktu: 44.02s)

Gambar 2.2.7.1 Perbandingan Hasil Model Buatan dengan MLP Sklearn

BAB III

KESIMPULAN DAN SARAN

A. Kesimpulan

Implementasi Feedforward Neural Network (FFNN) yang dikembangkan memenuhi spesifikasi yang ditetapkan seperti penerimaan konfigurasi jumlah neuron di tiap layer, berbagai fungsi aktivasi (Linear, ReLU, Sigmoid, Tanh, LeakyReLU, ELU, dan SoftMax), implementasi fungsi loss (MSE, Binary, dan Categorical Cross-Entropy), dan metode inisialisasi bobot (zero, random uniform, random normal, Xavier, dan He).

Proses forward propagation, backward propagation dengan penerapan chain rule, serta update parameter menggunakan gradient descent dijelaskan secara mendetail sehingga di akhir proses, model ini mampu menampilkan visualisasi struktur jaringan serta distribusi bobot dan gradien per layer. Hasil pengujian menunjukkan bahwa peningkatan depth dan width jaringan berdampak positif terhadap akurasi, pemilihan fungsi aktivasi yang tepat (LeakyReLU memberikan performa terbaik), serta penggunaan learning rate dan metode inisialisasi bobot yang optimal dapat meningkatkan performa model dalam hal ini learning rate sebesar 0,1 dan inisialisasi bobot he. Perbandingan dengan model MLP dari library sklearn juga menunjukkan bahwa meskipun akurasi kedua model serupa, model buatan ini memiliki keunggulan dari segi efisiensi waktu komputasi.

B. Saran

Untuk pengembangan lebih lanjut, disarankan agar ditambahkan fitur-fitur seperti automatic differentiation untuk mempermudah perhitungan gradien, eksplorasi fungsi aktivasi lainnya, dan optimizer alternatif seperti RMSprop atau SGD untuk meminimalkan fungsi loss. Percobaan lebih lanjut juga perlu mencakup pengujian pada dataset lain selain mnist_784 untuk menguji kemampuan generalisasi model.

PEMBAGIAN TUGAS

Nama / NIM	Tugas
Juan Alfred Widjaya / 13522073	Laporan, model, fungsi
Albert / 13522081	Laporan, model, testing
Ivan Hendrawan Tan / 13522111	Laporan, fungsi

REFERENSI

- [2020machinelearning.medium.com](https://2020machinelearning.medium.com/deep-dive-into-deep-learning-layers-rmsnorm-and-batch-normalization-b2423552be9f). (2024, 14 Maret). Deep Dive into Deep Learning: Layers, RMSNorm, and Batch Normalization. Diakses pada 19 Maret 2025, dari <https://2020machinelearning.medium.com/deep-dive-into-deep-learning-layers-rmsnorm-and-batch-normalization-b2423552be9f>
- [geeksforgeeks.org](https://www.geeksforgeeks.org/weight-initialization-techniques-for-deep-neural-networks). (2022, 04 Juli). Weight Initialization Techniques for Deep Neural Networks. Diakses pada 19 Maret 2025, dari <https://www.geeksforgeeks.org/weight-initialization-techniques-for-deep-neural-networks>
- [geeksforgeeks.org](https://www.geeksforgeeks.org/regularization-in-machine-learning). (2025, 03 Februari). Regularization in Machine Learning. Diakses pada 19 Maret 2025, dari <https://www.geeksforgeeks.org/regularization-in-machine-learning>
- [towardsdatascience.com](https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141). (2021, 21 September). Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis. Diakses pada 19 Maret 2025, dari <https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141>
- [v7labs.com](https://www.v7labs.com/blog/neural-networks-activation-functions). (2021, 27 Mei). Activation Functions in Neural Networks [12 Types & Use Cases]. Diakses pada 19 Maret 2025, dari <https://www.v7labs.com/blog/neural-networks-activation-functions>